

بِسْمِ اللّٰهِ الرَّحْمٰنِ الرَّحِيمِ

آموزش جامع برنامه نویسی با پایتون

دکتر مهدی رضوی فر

(عضو هیات علمی دانشگاه تبریز)

مهندس رضا سیدنژاد

(کارشناسی ارشد مهندسی شیمی دانشگاه تبریز)

Comprehensive Python Programming

Learning

Dr. Mehdi Razavifar

Faculty of chemical and Petroleum Engineering,
University of Tabriz

Mr. Reza Seyed Nezhad

Faculty of chemical and Petroleum Engineering,
University of Tabriz



پیشگفتار

آینده متعلق به هوش مصنوعی و دنیای دیجیتال است. در همین راستا، دانشجویان عزیز باید آماده حضوری پررنگ در این رقابت جهانی باشند. یادگیری برنامه نویسی و کدنویسی به عنوان مقدمه‌ای برای نقش آفرینی در دنیای مدرن و نوین آینده، بیش از هر زمان دیگری حائز اهمیت است. به همین منظور باید آموزش کدنویسی و برنامه نویسی با بیانی ساده و به صورت جامع در اختیار دانشجویان عزیز از تمام رشته‌ها قرار گیرد. در سال‌های اخیر برنامه نویسی با پایتون (Python) به دلیل مزایای آن از جمله سهولت در کدنویسی، امکان استفاده از آن بدون پرداخت هزینه، شی گرا بودن و داشتن کتابخانه‌های گسترده بیش از پیش مورد توجه قرار گرفته است و استفاده از پایتون نسبت به سایر زبان‌های برنامه نویسی با روند صعودی همراه بوده است.

در راستای نقش آفرینی موثر در آینده جهان و بسترسازی برای حضوری پررنگ در دنیای مدرن، نگارش کتاب آموزش پایتون به بیانی ساده انجام شد. در همین راستا، بخش‌های مهم و کلیدی از پایتون همراه با مثال‌های کاربردی در این کتاب ارائه شده‌اند. نگارش کتاب به نحوی انجام شده که افرادی که هیچ آشنایی با برنامه نویسی و کدنویسی در پایتون ندارند، هم امکان استفاده از این کتاب را داشته باشند و در انتهای، بتوانند برنامه‌های کاربردی مرتبط با رشته خود را کدنویسی و اجرا کنند. در واقع آموزش شروع آموزش برنامه نویسی با این کتاب به هیچ پیش نیازی وابسته نیست و تمام بخش‌ها به صورت جامع و دقیق همراه با مثال‌های موردی توضیح داده شده‌اند.

استفاده از کتاب پیش رو محدود به دانشجویان یا فارغ‌التحصیلان رشته‌ای خاص نیست و همه متقاضیان و علاقمندان به کدنویسی امکان استفاده از این کتاب را دارند. در این کتاب، ابتدا به ضرورت یادگیری کدنویسی و برنامه نویسی پرداخته شده و سپس مزایای پایتون نسبت به سایر زبان‌ها و نرم افزارهای کدنویسی دیگر ارائه شده است. در ادامه نحوه نصب پایتون و واسطه کاربری آن بیان شده و به توضیح قواعد و اصول کدنویسی در پایتون در پرداخته شده است. هم چنین کتابخانه‌های مهم این پایتون و نحوه نصب و کار با آن‌ها نیز به تفکیک ارائه شده‌اند. به ویژه کتابخانه‌های مرتبط با پروژه‌های آماری، محاسباتی، رسم نمودارها و معادلات ریاضی به تفضیل

بیان شده‌اند. امید است کتاب ارائه شده گامی مفید و موثر در جهت ارتقای سطح علمی دانشجویان و فارغ التحصیلان عزیز باشد و در نهایت منجر به سربلندی بیشتر کشور عزیزمان ایران شود. در انتها، از شما عزیزان خواهشمندیم، پیشنهادات و نظرات ارزشمند خود را با ما از طریق آدرس ایمیل اعلام شده مطرح کنید.

با تشکر

دکتر مهدی رضوی فر (عضو هیات علمی دانشگاه تبریز) (m.razavifar@tabrizu.ac.ir)

مهندس رضا سیدنژاد (کارشناسی ارشد مهندسی شیمی دانشگاه تبریز)

فهرست عناوین

۱) آشنایی با برنامهنویسی و زبان‌های برنامهنویسی.....	۴۷
۱-۱) اهمیت و نقش برنامهنویسی.....	۴۷
۲-۱) گام‌های اولیه در یادگیری برنامهنویسی.....	۴۸
۳-۱) خلاقیت در برنامهنویسی	۴۸
۴-۱) تعریف برنامهنویسی	۴۸
۵-۱) تاریخچه برنامهنویسی	۵۰
۶-۱) زبان‌های برنامهنویسی	۵۰
۱-۶-۱) زبان‌های برنامهنویسی سطح پایین	۵۱
۱-۶-۲) زبان‌های برنامهنویسی سطح بالا.....	۵۲
۱-۶-۳) آشنایی با زبان برنامهنویسی پایتون	۵۲
۱-۶-۴) آشنایی با زبان برنامهنویسی سی	۵۳
۱-۶-۵) آشنایی با زبان برنامهنویسی سی‌پلاس‌پلاس	۵۵
۱-۶-۶) آشنایی با زبان برنامهنویسی سی‌شارپ	۵۶
۱-۶-۷) آشنایی با زبان برنامهنویسی جاوا.....	۵۸
۱-۶-۸) آشنایی با زبان برنامهنویسی متلب	۵۹
۲) آشنایی با زبان برنامهنویسی پایتون و نصب و راهاندازی آن	۶۲
۲-۱) تاریخچه زبان برنامهنویسی پایتون	۶۳

۶۴	۲-۲) ویژگی‌های زبان برنامه‌نویسی پایتون
۶۶	۲-۳) کاربردهای زبان برنامه‌نویسی پایتون
۶۸	۴-۲) جایگاه زبان پایتون در شرکت‌های بزرگ دنیا
۶۹	۴-۳) نصب و راه اندازی پایتون
۷۱	۴-۵-۱) نصب روی سیستم عامل ویندوز
۷۳	۴-۵-۲) نصب روی سیستم عامل مک
۷۶	۴-۵-۳) معرفی نرم‌افزارهایی جهت کدنویسی پایتون
۸۰	۶-۲ آشنایی با محیط VS Code و نصب افزونه‌های پایتون
۸۵	۷-۲ اجرای کدهای پایتون با VS Code
۸۵	۳) آشنایی با مقدمات و ساختار زبان برنامه‌نویسی پایتون
۸۷	۳-۱) نحوه کامنت‌گذاری در پایتون
۸۹	۳-۲) آشنایی با تابع چاپ (print)
۸۹	۳-۳) آشنایی با f-string در پایتون
۹۱	۳-۳-۱) نحوه تعریف f-strings
۹۱	۳-۳-۲) ویژگی‌های کلیدی f-strings
۹۱	۳-۳-۳) مزایای f-strings
۹۱	۴-۳-۱) مثال مربوط به تابع print در f-string
۹۲	۴-۳-۲) انواع عملگرها در پایتون

۹۳.....	۱-۴-۳) عملگرهای ریاضی
۹۴.....	۲-۴-۳) عملگرهای مقایسه‌ای
۹۶.....	۳-۴-۳) عملگرهای تخصیص
۹۷.....	۴-۴-۳) عملگرهای منطقی
۱۰۰	۵-۴-۳) اولویت عملگرها
۱۰۱	۵) معرفی توابع وابسته در پایتون
۱۰۳.....	۴) متغیرها و انواع داده‌ها در زبان برنامه‌نویسی پایتون
۱۰۳.....	۱-۴) متغیرها در پایتون
۱۰۵.....	۲-۴) انواع داده‌ها در پایتون
۱۰۵.....	۱-۲-۴) اعداد در پایتون
۱۰۹.....	۲-۲-۴) رشته‌ها در پایتون
۱۳۲.....	۳-۲-۴) فهرست‌ها در پایتون
۱۵۱.....	۴-۲-۴) دیکشنری‌ها در پایتون
۱۷۱.....	۵-۲-۴) تاپل‌ها در پایتون
۱۸۶.....	۶-۲-۴) مجموعه‌ها در پایتون
۲۰۷.....	۵) دستورات شرطی در پایتون
۲۰۸.....	۱-۵) دستور شرطی if
۲۱۰	۱-۱-۵) ساختار if

۲۱۱ if/else ساختار ۵-۱-۲)
۲۱۳ if/elif/else ساختار ۵-۱-۳)
۲۱۶ مقایسه‌ی بین شرط‌ها در دستورات شرطی ۵-۱-۴)
۲۲۱ دستورات شرطی تو در تو ۵-۱-۵)
۲۲۶ ۶) حلقه‌ها در پایتون
۲۲۶ ۶-۱) حلقه while در پایتون
۲۲۷ ۶-۱-۱) مثال‌های مربوط به حلقه while
۲۳۲ ۶-۱-۲) بخش else در حلقه while
۲۳۵ ۶-۱-۳) دستور continue و break در حلقه while
۲۳۹ ۶-۲) حلقه for در پایتون
۲۴۱ ۶-۲-۱) تابع `range`
۲۴۲ ۶-۲-۲) بخش else در حلقه for
۲۴۵ ۶-۲-۳) مثال‌های مربوط به حلقه for
۲۵۲ ۶-۳) حلقه‌های تو در تو
۲۵۳ ۶-۳-۱) شیوه‌ی کار حلقه‌های تو در تو
۲۵۴ ۶-۳-۲) کاربردها و موارد استفاده حلقه‌های تو در تو
۲۵۶ ۷) مقدمه‌ای بر توابع در زبان برنامه‌نویسی پایتون
۲۵۶ ۷-۱) تعریف و هدف از توابع

۲۵۹	۷-۲) مزایای استفاده از توابع
۲۶۰	۷-۳) توابع داخلی و توابع تعریف شده توسط کاربر
۲۶۰	۷-۴) نحوه اجرا و فرآخوانی توابع
۲۶۱	۷-۴-۱) فرآخوانی تابع با پرانتز
۲۶۲	۷-۴-۲) فرآخوانی تابع بدون پرانتز
۲۶۳	۷-۵) پارامترها و آرگومانها در توابع
۲۶۴	۷-۵-۱) تعریف پارامترها
۲۶۹	۷-۵-۲) تعریف آرگومان
۲۷۲	۷-۵-۳) مثال و تفاوت میان پارامترها و آرگومانها
۲۷۳	۷-۵-۴) آرگومان های دلخواه
۲۸۰	۷-۶) توابع درونی در زبان برنامه نویسی پایتون
۲۸۵	۷-۷) آشنایی با lambda در زبان برنامه نویسی پایتون
۲۸۵	۷-۷-۱) مثال های مربوط به lambda
۲۹۲	۷-۸) آشنایی با توابع درونی map(), filter(), zip()
۲۹۲	۷-۸-۱) تابع درونی zip()
۲۹۴	۷-۸-۲) تابع درونی map()
۲۹۷	۷-۸-۳) تابع درونی filter()
۳۰۱	۸) خطاهای و انواع آنها در زبان برنامه نویسی پایتون

۳۰۱	۸-۱) خطاهای ساختاری.....
۳۰۲	۸-۲) خطاهای زمان اجرا.....
۳۰۴	۸-۳) خطاهای منطقی.....
۳۰۶	۸-۴) سایر خطاهای متداول در پایتون.....
۳۱۱	۹) مدیریت خطاهای در پایتون.....
۳۱۱	۹-۱) ساختار try/except برای مدیریت کردن خطاهای.....
۳۱۳	۹-۲) مدیریت چندین نوع خطای.....
۳۱۴	۹-۳) استفاده از بلوک عمومی except.....
۳۱۵	۹-۴) ترکیب با else و finally.....
۳۱۶	۹-۵) کاربرد مدیریت استثنایها.....
۳۱۸	۱۰) مفهوم دامنه در زبان برنامه نویسی پایتون.....
۳۱۸	۱۰-۱) انواع Scope در پایتون.....
۳۱۸	۱۰-۱-۱) دامنه محلی.....
۳۱۹	۱۰-۱-۲) دامنه بسته.....
۳۲۰	۱۰-۱-۳) دامنه سراسری.....
۳۲۱	۱۰-۱-۴) دامنه داخلی.....
۳۲۱	۱۰-۲) قانون جستجوی نامها یا قانون LEGB.....
۳۲۲	۱۰-۳) متغیرهای سراسری و کلیدواژه global.....

۳۲۳.....	۴-۱۰) متغیرهای محلی و کلیدواژه nonlocal
۳۲۶.....	۱۱) برنامه‌نویسی شی‌گرا
۳۲۷.....	۱-۱۱) کلاس و شی در زبان برنامه‌نویسی پایتون
۳۲۸.....	۱-۱-۱۱) کلاس‌ها در پایتون
۳۲۸.....	۲-۱-۱۱) شیء در پایتون
۳۲۸.....	۱-۱-۳) ویژگی‌ها و رفتارهای کلاس و اشیاء
۳۲۹.....	۴-۱-۱۱) مثال‌هایی از کلاس و شیء
۳۲۹.....	۱-۱-۵) مزایای استفاده از کلاس و شیء در پایتون
۳۳۰.....	۱-۱-۶) تعریف کلاس و شیء در پایتون
۳۳۱.....	۷-۱-۱۱) تابع __init__()
۳۳۷.....	۱-۱-۸) تابع __str__()
۳۳۸.....	۱-۱-۹) تابع __repr__()
۳۴۱.....	۱۰-۱-۱۱) تفاوت بین __str__() و __repr__()
۳۴۲.....	۱۱-۱-۱۱) توابع وابسته در کلاس‌های پایتون
۳۴۸.....	۲-۱-۱۱) وراثت در پایتون
۳۴۸.....	۱-۲-۱۱) ویژگی‌های کلیدی وراثت
۳۴۹.....	۲-۲-۱۱) نحوه استفاده از وراثت در پایتون
۳۵۰.....	۳-۲-۱۱) مثالی از وراثت تک‌گانه

۳۵۱	تابع وابسته‌ی super() ۴-۲-۱۱
۳۵۳	وراثت چندگانه در پایتون ۵-۲-۱۱
۳۵۵	وراثت چندسطحی در پایتون ۶-۲-۱۱
۳۵۶	چندریختی در پایتون ۳-۱-۱۱
۳۵۶	۱) انواع چندریختی در پایتون ۳-۳-۱۱
۳۵۷	۲) ویژگی‌های چندریختی ۳-۳-۱۱
۳۵۷	۳) ترکیب چندریختی با ارثبری و Duck Typing ۳-۳-۱۱
۳۶۰	۴) مزایای چندریختی ۴-۳-۱۱
۳۶۲	۱۲) مازول‌ها در پایتون ۱۲
۳۶۲	۱) چرا باید از مازول‌ها استفاده کنید? ۱-۱۲
۳۶۲	۲) انواع مازول‌ها ۲-۱۲
۳۶۲	۱) مازول‌های داخلی ۱-۲-۱۲
۳۶۳	۲) مازول‌های تعریف شده توسط کاربر ۲-۲-۱۲
۳۶۴	۳) مازول‌های شخص ثالث ۲-۲-۱۲
۳۶۴	۳) وارد کردن مازول‌ها به داخل پروژه ۳-۱-۱۲
۳۶۴	۱) وارد کردن پایه‌ای ۱-۳-۱۲
۳۶۵	۲) وارد کردن مقدار خاص مانند کلاس از یک مازول ۲-۳-۱۲
۳۶۵	۳) وارد کردن تمامی مقادیری که در داخل مازول وجود دارد ۳-۳-۱۲

۳۶۵	وارد کردن مازول با دادن نام مستعار برای آن مازول	۴-۳-۱۲
۳۶۶	ایجاد و استفاده از مازول	۴-۱۲
۳۶۷	متغیر <code>name</code> در پایتون	۵-۱۲
۳۶۸	<code>name</code> عملکرد	۱-۵-۱۲
۳۶۸	<code>name</code> کاربرد	۲-۵-۱۲
۳۶۸	<code>name</code> مثال‌های مربوط به	۳-۵-۱۲
۳۷۱	<code>name</code> مزیای استفاده از	۴-۵-۱۲
۳۷۳	آشنایی با مازول Random در پایتون	۱۳
۳۷۳	random ویژگی‌های مازول	۱-۱۳
۳۷۴	random محدودیت‌های مازول	۲-۱۳
۳۷۵	random توابع وابسته در مازول	۳-۱۳
۳۷۶	random کاربردهای اصلی و توابع مهم مازول	۴-۱۳
۳۷۶	random تولید اعداد تصادفی با استفاده از مازول	۱-۴-۱۳
۳۷۸	random تولید اعداد صحیح	۲-۴-۱۳
۳۷۹	کار با فهرست‌ها	۳-۴-۱۳
۳۸۳	random توابع توزیع آماری	۴-۴-۱۳
۳۸۸	آشنایی با مازول math در پایتون	۱۴
۳۸۸	math ویژگی‌های مازول	۱-۱۴

۳۸۹	۲-۱۴) کاربردهای ماژول math
۳۹۲	۳-۱۴) توابع وابسته موجود در ماژول math
۳۹۳	۴-۱۴) توابع مهم و پرکاربرد ماژول math
۳۹۴	۱-۴-۱۴) محاسبات عددی
۳۹۷	۲-۴-۱۴) توابع مثلثاتی
۴۰۲	۳-۴-۱۴) توابع نمایی و لگاریتمها
۴۰۵	۴-۴-۱۴) مدیریت اعداد اعشاری و صحیح
۴۱۰	۵-۴-۱۴) ثابت‌های ریاضی
۴۱۳	۵-۱۴) مثال‌هایی از کاربردهای عملی
۴۱۵	۱۵) آشنایی با ماژول Datetime
۴۱۶	۱-۱۵) ویژگی‌های ماژول datetime
۴۱۸	۲-۱۵) کاربردهای ماژول datetime
۴۱۹	۳-۱۵) توابع وابستهٔ موجود در ماژول Datetime
۴۲۱	۴-۱۵) توابع مهم و پرکاربرد ماژول Datetime
۴۲۱	۱-۴-۱۵) برگرداندن تاریخ و زمان فعلی سیستم
۴۲۲	۲-۴-۱۵) برگرداندن تاریخ و زمان فعلی سیستم بدون تنظیمات منطقه زمانی
۴۲۳	۳-۴-۱۵) تبدیل یک رشته تاریخ‌زمان به یک شیء datetime
۴۲۴	۴-۴-۱۵) تبدیل یک شیء datetime را به رشته تاریخ‌زمان

۴۲۴.....	۵-۴) برگرداندن تاریخ و زمان معادل یک زمان یونیکس.....	۱۵
۴۲۵.....	۶-۴) ترکیب یک تاریخ و زمان و ایجاد یک شیء <code>datetime</code>	۱۵
۴۲۶.....	۷-۴) تغییر دادن قسمتی از یک شیء <code>datetime</code>	۱۵
۴۲۷.....	۸-۴) برگرداندن زمان یونیکس معادل یک شیء <code>datetime</code>	۱۵
۴۲۹.....	(۱) آشنایی با ماژول نامپای در پایتون.....	۱۶
۴۲۹.....	(۱-۱) ویژگی‌های اصلی نامپای.....	۱۶
۴۳۰.....	(۲) کاربردهای نامپای.....	۱۶
۴۳۰.....	(۳) مزایای نامپای.....	۱۶
۴۳۱.....	(۴) مقایسه با فهرست‌ها در پایتون.....	۱۶
۴۳۱.....	(۵) نصب نامپای.....	۱۶
۴۳۱.....	(۱-۵) نصب ماژول نامپای با <code>pip</code>	۱۶
۴۳۱.....	(۲-۵) بررسی نصب ماژول نامپای.....	۱۶
۴۳۲.....	(۳-۵) نصب نسخه خاص از ماژول نامپای.....	۱۶
۴۳۲.....	(۴-۵) بروزرسانی ماژول نامپای.....	۱۶
۴۳۲.....	(۵-۵) نصب ماژول نامپای در محیط مجازی.....	۱۶
۴۳۳.....	(۶-۵) ماتریس‌ها در نامپای.....	۱۶
۴۳۳.....	(۱-۶) ویژگی‌های ماتریس‌ها در نامپای.....	۱۶
۴۳۳.....	(۲-۶) ایجاد ماتریس در ماژول نامپای.....	۱۶

۴۳۷	۳-۶) عملیات‌های اصلی ماتریسی ۱۶
۴۴۱	۴-۶) معکوس ماتریس ۱۶
۴۴۲	۵-۶) دترمینان ماتریس ۱۶
۴۴۳	۶-۶) دسترسی به عناصر ماتریس ۱۶
۴۴۵	۶-۸) تغییر شکل ماتریس: ۱۶
۴۴۶	۷-۶) مثال‌های کاربردی نامپای ۱۶
۴۴۶	۷-۷-۱) ایجاد آرایه ۱۶
۴۴۷	۷-۷-۲) ایجاد آرایه‌های خاص ۱۶
۴۴۹	۷-۷-۳) شکل و اندازه آرایه ۱۶
۴۵۰	۷-۴) عملیات عددی ۱۶
۴۵۱	۷-۵) توابع ریاضی ۱۶
۴۵۴	۱۷) آشنایی با ماژول SymPy در پایتون ۱۷
۴۵۴	۱-۱۷) ویژگی‌های SymPy ۱۷
۴۵۵	۲-۱۷) کاربردهای SymPy ۱۷
۴۵۶	۳-۱۷) نقاط قوت SymPy ۱۷
۴۵۷	۴-۱۷) نصب و وارد کردن ماژول SymPy ۱۷
۴۵۷	۵-۱۷) ایجاد متغیرهای نمادین در ماژول SymPy ۱۷
۴۵۸	۱-۵-۱۷) تعریف متغیرهای نمادین ۱۷

۴۵۸	۲-۵-۱۷) ایجاد متغیر نمادین با symbols
۴۵۸	۳-۵-۱۷) ایجاد متغیر با ویژگی‌های خاص در مازول SymPy
۴۵۹	۴-۵-۱۷) ایجاد متغیر با Symbol
۴۵۹	۵-۵-۱۷) ایجاد آرایه‌ای از متغیرها
۴۶۰	۶-۱۷) سادهسازی و گسترش عبارات
۴۶۰	۱-۶-۱۷) سادهسازی عبارات در مازول SymPy
۴۶۲	۷-۱۷) مبحث حد در مازول SymPy
۴۶۲	۱-۷-۱۷) محاسبه حد با استفاده از مازول SymPy
۴۶۲	۲-۷-۱۷) محاسبه حد در نقاط مشخص با استفاده از مازول SymPy
۴۶۳	۳-۷-۱۷) محاسبه حد در بینهایت با استفاده از مازول SymPy
۴۶۴	۴-۷-۱۷) محاسبه حد از چپ و راست با استفاده از مازول SymPy
۴۶۶	۵-۷-۱۷) حد توابع مثلثاتی با استفاده از مازول SymPy
۴۶۷	۶-۷-۱۷) بررسی حد برای نقاط ناپیوسته با استفاده از مازول SymPy
۴۶۸	۸-۱۷) مبحث مشتق در مازول SymPy
۴۶۹	۱-۸-۱۷) محاسبه مشتق با استفاده از مازول SymPy
۴۶۹	۲-۸-۱۷) محاسبه مشتق اول با استفاده از مازول SymPy
۴۷۰	۳-۸-۱۷) محاسبه مشتق بالاتر با استفاده از مازول SymPy
۴۷۱	۴-۸-۱۷) محاسبه مشتق توابع چندمتغیره با استفاده از مازول SymPy

۴۷۲ ۵-۸) محاسبه مشتق توابع مثلثاتی با استفاده از ماژول SymPy
۴۷۳ ۶-۸-۱۷) محاسبه مشتق توابع مرکب با استفاده از ماژول SymPy
۴۷۴ ۹-۱۷) مبحث انتگرال در ماژول SymPy
۴۷۴ ۹-۱۷) تعریف انتگرال نامعین
۴۷۴ ۲-۹-۱۷) تعریف انتگرال معین
۴۷۴ ۳-۹-۱۷) محاسبه انتگرال در SymPy
۴۷۵ ۴-۹-۱۷) محاسبه انتگرال نامعین در ماژول SymPy
۴۷۵ ۵-۹-۱۷) محاسبه انتگرال معین در ماژول SymPy
۴۷۶ ۶-۹-۱۷) محاسبه انتگرال توابع مثلثاتی در ماژول SymPy
۴۷۷ ۷-۹-۱۷) محاسبه انتگرال توابع چند متغیره در ماژول SymPy
۴۷۸ ۸-۹-۱۷) محاسبه انتگرال عددی در ماژول SymPy
۴۷۹ ۹-۹-۱۷) محاسبه انتگرال توابع پیچیده در ماژول SymPy
۴۸۱ ۱۰-۱۷) حل معادلات جبری با استفاده از ماژول SymPy
۴۸۱ ۱۰-۱۷) توابع حل معادلات در ماژول SymPy
۴۸۲ ۱۰-۱۷) ساختار کلی استفاده ازتابع solve در ماژول SymPy
۴۸۲ ۱۰-۱۷) حل معادلات ساده با استفاده از ماژول Sympy
۴۸۴ ۱۰-۱۷) حل معادلات چندمتغیره با استفاده از ماژول Sympy
۴۸۵ ۱۰-۱۷) حل معادلات غیرخطی با استفاده از ماژول Sympy

۴۸۶ حل معادلات با solveset (۱۰-۶)
۴۸۷ (۱) حل معادلات دیفرانسیل با استفاده از مازول SymPy (۱۱-۱۷)
۴۸۷ (۲) تعریف معادله دیفرانسیل در SymPy (۱۱-۱۱-۱)
۴۸۷ (۳) توابع اصلی برای حل معادلات دیفرانسیل (۱۱-۱۱-۲)
۴۸۸ (۴) حل معادلات دیفرانسیل ساده (۱۱-۱۱-۳)
۴۸۹ (۵) معادلات دیفرانسیل غیرخطی (۱۱-۱۱-۴)
۴۹۰ (۱) مثال‌های کاربردی SymPy (۱۲-۱۷)
۴۹۰ (۲) ساده‌سازی عبارات ریاضی (۱۲-۱۲-۱)
۴۹۱ (۳) حل معادلات جبری (حل معادله درجه دوم) (۱۲-۱۲-۲)
۴۹۲ (۴) محاسبه مشتق یک تابع (۱۲-۱۲-۳)
۴۹۳ (۵) انتگرال‌گیری از یک تابع (۱۲-۱۲-۴)
۴۹۴ (۶) سری تیلور (۱۲-۱۲-۵)
۴۹۵ (۷) کاربرد در هندسه (محاسبه مساحت دایره) (۱۲-۱۲-۶)
۴۹۸ (۱) آشنایی با مازول pandas در پایتون (۱۸)
۴۹۸ (۲) ویژگی‌های اصلی Pandas (۱۸-۱-۱)
۵۰۰ (۳) کاربردهای Pandas (۱۸-۲)
۵۰۱ (۴) مزایای استفاده از Pandas (۱۸-۳)

۵۰۱.....	۱-۳) مقایسه Pandas با سایر مازول‌ها
۵۰۲.....	۴-۱۸) نصب و وارد کردن مازول Pandas به پروژه
۵۰۲.....	۵-۱۸) توابع وابسته‌ی کاربردی در مازول Pandas
۵۰۲.....	۵-۵-۱) بارگذاری داده‌ها از فایل CSV و نمایش داده‌ها
۵۰۳.....	۲-۵-۱) تمیز کردن داده‌ها (شناسایی و حذف داده‌های گمشده)
۵۰۳.....	۳-۵-۱) مرتبسازی داده‌ها (مرتبسازی داده‌ها بر اساس یک ستون)
۵۰۴.....	۴-۵-۱) گروهبندی داده‌ها (محاسبه مجموع داده‌ها در گروه‌های مختلف)
۵۰۶.....	۵-۵-۱) محاسبات آماری (محاسبه میانگین، کمینه و بیشینه داده‌ها)
۵۰۶.....	۶-۵-۱) انتخاب داده‌ها (فیلتر کردن داده‌ها بر اساس یک شرط)
۵۰۶.....	۷-۵-۱) ایجاد ستون جدید (ایجاد یک ستون جدید بر اساس محاسبات ستون‌های موجود)
۵۰۶.....	۸-۵-۱) ترکیب و ادغام داده‌ها
۵۰۷.....	۹-۵-۱) تغییر فرمت داده‌ها (تبديل تاریخ‌ها به فرمت زمانی)
۵۰۷.....	۱۰-۵-۱) ذخیره داده‌ها (ذخیره داده‌های ویرایش شده به یک فایل جدید)
۵۰۷.....	۱۱-۵-۱) محاسبه درصدها (محاسبه نسبت درصدی یک ستون)
۵۰۷.....	۱۲-۵-۱) حذف ستون‌ها
۵۰۷.....	۱۳-۵-۱) شناسایی و حذف داده‌های تکراری
۵۰۸.....	۶-۱۸) پروژه مربوط به مازول pandas
۵۱۰.....	۱۹) آشنایی با مازول Matplotlib در پایتون

۵۱۰	Matplotlib ویژگی‌های (۱-۱۹)
۵۱۱	Matplotlib زیرماژول‌های (۲-۱۹)
۵۱۱	Matplotlib کاربردهای (۳-۱۹)
۵۱۲	ترسیم نمودارها با استفاده از ماژول Matplotlib در پایتون (۴-۱۹)
۵۱۲	(۱) ترسیم نمودار خطی (۴-۱۹)
۵۱۳	(۲) ترسیم نمودار میله‌ای (۴-۱۹)
۵۱۵	(۳) ترسیم نمودار دایره‌ای (۴-۱۹)
۵۱۶	(۴) ترسیم نمودار پراکندگی (۴-۱۹)
۵۱۷	(۵) ترسیم هیستوگرام (۴-۱۹)
۵۱۸	(۶) ترسیم چند نمودار در یک صفحه (۴-۱۹)
۵۲۰	(۷) ترسیم نمودار سه‌بعدی (۴-۱۹)
۵۲۲	(۸) ترسیم نمودارهای log-log در ماژول matplotlib (۴-۱۹)
۵۲۹	(۹) ترسیم نمودارهای Semi-log در ماژول matplotlib (۴-۱۹)
۵۳۶	Matplotlib (۵-۱۹)
۵۳۶	(۱) برچسب‌گذاری محورها (۵-۱۹)
۵۳۸	(۲) تنظیم محدوده محورها (۵-۱۹)
۵۳۹	(۳) اضافه کردن شبکه به محورها (۵-۱۹)
۵۴۱	(۴) برچسب‌های دلخواه برای محورها (۵-۱۹)

۵۴۲	۱-۵) چرخش برچسبهای محورها.....
۵۴۴	۲-۵) افزودن خطوط به محورها.....
۵۴۵	۳-۶) ایجاد چندین نمودار در یک صفحه.....
۵۴۵	۴-۶) ساختار کلی subplot()
۵۴۶	۵-۶) رسم چند نمودار ساده.....
۵۴۷	۶-۶) چند نمودار در چند ردیف و ستون
۵۴۹	۷-۶) استفاده از Axes و Figure
۵۵۱	۷-۱) نمودارهای هیستوگرام و میلهای در مازول Matplotlib
۵۵۱	۷-۲) نمودار هیستوگرام
۵۵۴	۷-۳) نمودار میلهای
۵۵۶	۷-۴) تفاوت‌های اصلی بین هیستوگرام و میلهای
۵۵۶	۷-۵) ترکیب نمودارهای هیستوگرام و میلهای
۵۵۹	منابع.....

فهرست اشکال

فصل دوم

..... شکل ۲ - ۱ - صفحه‌ی اصلی وبگاه رسمی پایتون	۴۶
..... شکل ۲ - ۲ - صفحه‌ی اصلی فایل پایتون دانلود شده از وبگاه پایتون	۴۷
..... شکل ۲ - ۳ - صفحه‌ی مربوط به موفقیت آمیز بودن نصب پایتون	۴۷
..... شکل ۲ - ۴ - صفحه‌ی اصلی وبگاه رسمی VS Code	۵۲
..... شکل ۲ - ۵ - محیط کلی از نرمافزار VS Code	۵۳
..... شکل ۲ - ۶ - آیکون مربوط به قسمت Explorer در نرمافزار VS Code	۵۴
..... شکل ۲ - ۷ - آیکون مربوط به قسمت Search در نرمافزار VS Code	۷۸
..... شکل ۲ - ۸ - آیکون مربوط به قسمت Extensions در نرمافزار VS Code	۷۹
..... شکل ۲ - ۹ - نصب افزونه Python از قسمت Extension در VS Code	۷۹
..... شکل ۲ - ۱۰ - نصب افزونه Pylance از قسمت Extension در VS Code	۸۰
..... شکل ۲ - ۱۱ - نصب افزونه Python Indent از قسمت Extension در VS Code	۸۰
..... شکل ۲ - ۱۲ - نحوه‌ی ایجاد فایل در VS Code	۸۱
..... شکل ۲ - ۱۳ - فایل پایتونی ایجاد شده در VS Code	۸۱
..... شکل ۲ - ۱۴ - نوشتن برنامه چاپ "Hello World!" در VS Code	۸۲
..... شکل ۲ - ۱۵ - نحوه‌ی خروجی گرفتن از کد در ترمینال در VS Code	۸۳
..... شکل ۲ - ۱۶ - خروجی برنامه چاپ "Hello World!" در ترمینال در VS Code	۸۳

فصل سوم

شکل ۳-۱- نحوه کامنت‌گذاری تک خطی و چندخطی در پایتون	۸۶
شکل ۳-۲- مثالی از پارامتر <code>sep</code> در تابع <code>print</code>	۸۸
شکل ۳-۳- مثالی از پارامتر <code>end</code> در تابع <code>print</code>	۸۸
شکل ۳-۴- مثالی از <code>f-string</code> در تابع <code>print</code>	۹۲
شکل ۳-۵- مثالی از عملگر منطقی <code>and</code> در پایتون	۹۸
شکل ۳-۶- مثالی از عملگر منطقی <code>or</code> در پایتون	۱۰۰

فصل چهارم

شکل ۴-۱- مثالی از دستور <code>type</code> در پایتون	۱۰۵
شکل ۴-۲- مثالی از نوع عددی <code>int</code> در پایتون	۱۰۷
شکل ۴-۳- مثالی از نوع عددی <code>float</code> در پایتون	۱۰۸
شکل ۴-۴- مثالی از نوع عددی <code>complex</code> در پایتون	۱۰۹
شکل ۴-۵- تعریف رشته‌ها در پایتون	۱۱۰
شکل ۴-۶- مثالی از مفهوم <code>concentration</code> در پایتون	۱۱۷
شکل ۴-۷- مثالی از مفهوم <code>indexing</code> در پایتون	۱۱۸
شکل ۴-۸- مثالی از مفهوم <code>slicing</code> در پایتون	۱۲۰
شکل ۴-۹- مثالی از توابع وابسته‌ی <code>Lower</code> , <code>Upper</code> , <code>Capitalize</code>	۱۲۱
شکل ۴-۱۰- مثالی از توابع وابسته‌ی <code>Swapcase</code> , <code>casifold</code> , <code>title</code>	۱۲۳
شکل ۴-۱۱- مثالی از تابع وابسته <code>strip</code>	۱۲۴
شکل ۴-۱۲- مثالی از تابع وابسته <code>replace</code>	۱۲۶

- شکل ۴-۱۳- مثالی از تابع وابسته `split` ۱۲۷
- شکل ۴-۱۴- مثالی از تابع وابسته `find` ۱۲۹
- شکل ۴-۱۵- مثالی از تابع وابسته `index` ۱۳۰
- شکل ۴-۱۶- مثالی از توابع وابسته `index` و `find` با اعمال مقادیر دوم و سوم هر تابع وابسته ۱۳۱
- شکل ۴-۱۷- مثال‌هایی از ساختار فهرست‌ها در پایتون ۱۳۳
- شکل ۴-۱۸- به دست آوردن تعداد اعضای فهرست با استفاده از تابع `len` ۱۳۶
- شکل ۴-۱۹- دسترسی به یک عنصر خاص فهرست ۱۳۷
- شکل ۴-۲۰- دسترسی به محدوده خاص از فهرست ۱۳۸
- شکل ۴-۲۱- مثالی از تغییر یکی از عناصر فهرست ۱۳۹
- شکل ۴-۲۲- تغییر چند عنصر از فهرست ۱۴۰
- شکل ۴-۲۳- افروzen یک عنصر جدید به فهرست با استفاده از توابع وابسته `append` و `insert` ۱۴۱
- شکل ۴-۲۴- افروزن عناصر جدید به فهرست با تابع وابسته `extend` ۱۴۲
- شکل ۴-۲۵- حذف عنصر با توابع وابسته `pop` و `remove` و کلیدواژه `del` ۱۴۳
- شکل ۴-۲۶- حذف تمامی عناصر با تابع وابسته `clear` ۱۴۴
- شکل ۴-۲۷- مثالی از کپی‌گیری از فهرست با روش تابع وابسته `copy` و تابع داخلی `list` ۱۴۵
- شکل ۴-۲۸- مثالی از فهرست‌های تو در تو در پایتون ۱۴۶
- شکل ۴-۲۹- مثالی از دسترسی به عناصر فهرست‌های تو در تو در پایتون ۱۴۷
- شکل ۴-۳۰- مثالی از افزودن یک یا چند عنصر به داخل فهرست‌های تو در تو ۱۴۸
- شکل ۴-۳۱- مثالی از حذف عنصر از داخل فهرست‌های تو در تو ۱۴۹
- شکل ۴-۳۲- ادغام کردن دو فهرست با حلقه `for` و تابع وابسته `extend` ۱۵۰
- شکل ۴-۳۳- مثال‌هایی از دیکشنری‌ها در پایتون ۱۵۴

شکل ۴-۳۴- تعریف تابع داخل دیکشنری و استفاده از تابع ۱۵۵	
شکل ۴-۳۵- مثالی از تعریف کلید و مقدار برای دیکشنری در پایتون ۱۵۹	
شکل ۴-۳۶- مثالی از تغییر مقدار یک کلید دیکشنری ۱۶۰	
شکل ۴-۳۷- مثالی از حذف کلید و مقدار از دیکشنری با استفاده از توابع وابسته <code>pop</code> و <code>popitem</code> ۱۶۲	کلیدواژه <code>del</code>
شکل ۴-۳۸- مثالی از حذف تمامی کلید و مقدارها با استفاده از تابع وابسته <code>clear</code> در دیکشنری ۱۶۳	
شکل ۴-۳۹- مثالی از کپی‌گیری از دیکشنری با تابع وابسته <code>copy</code> و تابع داخلی <code>dict</code> ۱۶۴	
شکل ۴-۴۰- مثالی از دیکشنری‌های تو در تو در پایتون ۱۶۵	
شکل ۴-۴۱- مثالی از دسترسی به مقادیر در دیکشنری‌های تو در تو ۱۶۷	
شکل ۴-۴۲- تعریف کلید و مدار در دیکشنری‌های تو در تو ۱۶۸	
شکل ۴-۴۳- حذف کلید و مقدار در دیکشنری‌های تو در تو ۱۷۰	
شکل ۴-۴۴- تعریف و ایجاد تاپل‌ها در پایتون ۱۷۳	
شکل ۴-۴۵- مثالی از قرار دادن انواع داده‌ها در تاپل‌ها ۱۷۵	
شکل ۴-۴۶- به دست آوردن طول یک تاپل با استفاده از تابع <code>len</code> ۱۷۶	
شکل ۴-۴۷- مثالی از تاپل تک عنصری ۱۷۷	
شکل ۴-۴۸- مثالی از دسترسی به عناصر یک تاپل ۱۷۹	
شکل ۴-۴۹- مثالی از اندیس منفی در تاپل‌ها ۱۸۰	
شکل ۴-۵۰- مثال‌هایی از دسترسی به محدوده مشخص از تاپل ۱۸۱	
شکل ۴-۵۱- مثال‌هایی از بررسی موجود بودن یک عنصر در داخل یک تاپل ۱۸۲	
شکل ۴-۵۲- مثالی از تغییر عناصر تاپل با روش تبدیل تاپل به فهرست ۱۸۴	
شکل ۴-۵۳- مثالی از اضافه کردن یک تاپل به تاپل دیگر ۱۸۵	

..... ۱۸۷ شکل ۴-۵۴- مثالی از روش‌های تعریف مجموعه در پایتون
..... ۱۸۸ شکل ۴-۵۵- نوع داده‌های مجاز برای استفاده در داخل مجموعه‌ها
..... ۱۹۰ شکل ۴-۵۶- مثالی از به دست آوردن تعداد عناصر یک مجموعه با استفاده از تابع <code>len</code>
..... ۱۹۵ شکل ۴-۵۷- مثالی از وجود داشتن یا نداشتن یک عنصر در داخل یک مجموعه
..... ۱۹۷ شکل ۴-۵۸- مثالی از افزودن یک یا چند عنصر به داخل یک مجموعه
..... ۱۹۸ شکل ۴-۵۹- مثالی از حذف یک عنصر از مجموعه با استفاده از توابع وابسته <code>pop</code> و <code>discard</code> و <code>remove</code>
..... ۱۹۹ شکل ۴-۶۰- مثالی از حذف تمامی عناصر مجموعه با استفاده از تابع وابسته <code>clear</code>
..... ۲۰۰ شکل ۴-۶۱- مثالی از استفاده از توابع وابسته <code>union</code> و <code>update</code>
..... ۲۰۱ شکل ۴-۶۲- استفاده از توابع وابسته <code>intersection</code> و <code>difference</code> و <code>symmetric_difference</code>
..... ۲۰۳ شکل ۴-۶۳- مثال مربوط به نکات توابع وابسته <code>union</code> و <code>intersection</code>
..... ۲۰۴ شکل ۴-۶۴- مثال مربوط به نکات توابع وابسته <code>difference</code> و <code>symmetric_difference</code>
..... ۲۰۵ شکل ۴-۶۵- مثال مربوط به نکات مقادیر <code>True</code> و <code>False</code> در داخل مجموعه‌ها

فصل پنجم

..... ۲۰۹ شکل ۵-۱- فلوچارت مربوط به دستورات شرطی
..... ۲۱۱ شکل ۵-۲- مثالی از تعریف عبارات شرطی با <code>if</code>
..... ۲۱۲ شکل ۵-۳- مثالی از تعریف عبارات شرطی با <code>if/else</code>
..... ۲۱۴ شکل ۵-۴- مثالی از تعریف عبارات شرطی <code>if/elif/else</code> با دو شرط
..... ۲۱۶ شکل ۵-۵- مثالی از تعریف عبارات شرطی <code>if/elif/else</code> با بیش از دو شرط

..... ۱۹۷	شکل ۵-۶- مثال‌هایی از استفاده عملگرهای مقایسه‌ای بین دو یا چند شرط در دستورات شرطی.....
..... ۲۱۸	شکل ۵-۷- مثالی از استفاده <code>or and</code> در عبارات شرطی بین دو شرط.....
..... ۲۱۹	شکل ۵-۸- مثالی از کلیدواژه <code>in</code> در داخل عبارات شرطی.....
..... ۲۲۰	شکل ۵-۹- مثالی از کلیدواژه <code>not</code> در عبارات شرطی.....
..... ۲۲۲	شکل ۵-۱۰- مثال خرید خودرو برای کاربر اول با استفاده از عبارات شرطی <code>تو در تو</code>
..... ۲۲۳	شکل ۵-۱۱- مثال خرید خودرو برای کاربر دوم با استفاده از عبارات شرطی <code>تو در تو</code>
..... ۲۲۴	شکل ۵-۱۲- مثال خرید خودرو برای کاربر سوم با استفاده از عبارات شرطی <code>تو در تو</code>

فصل ششم

..... ۲۲۷	شکل ۶-۱- مثال چاپ اعداد ۱ تا ۵ با استفاده از حلقه <code>while</code>
..... ۲۲۹	شکل ۶-۲- مثال مربوط به شمارش معکوس با استفاده از حلقه <code>while</code>
..... ۲۳۱	شکل ۶-۳- مثال مربوط به محاسبه فاکتوریل یک عدد با استفاده از حلقه‌ی <code>while</code>
..... ۲۳۴	شکل ۶-۴- مثال مربوط به بخش <code>else</code> در حلقه <code>while</code>
..... ۲۳۶	شکل ۶-۵- مثال مربوط به کلیدواژه <code>break</code> در حلقه‌ی <code>while</code>
..... ۲۳۸	شکل ۶-۶- مثال مربوط به کلیدواژه <code>continue</code> در حلقه‌ی <code>while</code>
..... ۲۴۰	شکل ۶-۷- مثال مربوط به چاپ اعداد ۱ تا ۵ با استفاده از حلقه <code>for</code>
..... ۲۴۱	شکل ۶-۸- مثال مربوط به استفاده از تابع <code>range</code> در حلقه <code>for</code>
..... ۲۴۴	شکل ۶-۹- مثال مربوط به استفاده از بخش <code>else</code> در حلقه <code>for</code>
..... ۲۴۵	شکل ۶-۱۰- مثال مربوط به محاسبه میانگین نمرات یک دانشجو با استفاده از حلقه <code>for</code>
..... ۲۴۷	شکل ۶-۱۱- مثال مربوط به جداسازی اعداد زوج و فرد با استفاده از حلقه <code>for</code>

شکل ۶-۱۲- مثال مربوط به جداسازی حروف یک رشته با استفاده از حلقه <code>for</code>	۲۴۸
شکل ۶-۱۳- مثال مربوط به جداسازی کلید و مقدار یک دیکشنری با استفاده از حلقه <code>for</code>	۲۵۰
شکل ۶-۱۴- مثال مربوط به محاسبه حاصل ضرب اعداد یک فهرست با استفاده از حلقه <code>for</code>	۲۵۱
شکل ۶-۱۵- مثال حلقه‌های تو در تو	۲۵۳

فصل هفتم

شکل ۷-۱- مثال چاپ <code>Hello, World!</code> با استفاده از تعریف تابع در پایتون	۲۵۷
شکل ۷-۲- مثال محاسبه مجموع اعداد داخل یک فهرست با استفاده از تعریف تابع	۲۵۸
شکل ۷-۳- مثال محاسبه فاکتوریل یک عدد با استفاده از تعریف تابع	۲۵۹
شکل ۷-۴- مثال مربوط به فراخوانی تابع با پرانتز در پایتون	۲۶۱
شکل ۷-۵- مثال مربوط به فراخوانی تابع بدون پرانتز در پایتون	۲۶۲
شکل ۷-۶- مثالی از تعریف پارامتر برای تابع در پایتون	۲۶۴
شکل ۷-۷- مثالی از تعریف پارامتر موقعیتی در توابع پایتون	۲۶۵
شکل ۷-۸- مثالی از تعریف پارامتر کلیدواژه‌ای در توابع پایتون	۲۶۶
شکل ۷-۹- مثالی از تعریف پارامتر پیشفرض در توابع پایتون	۲۶۷
شکل ۷-۱۰- مثالی از تعریف پارامتر دلخواه در توابع پایتون	۲۶۸
شکل ۷-۱۱- مثالی از تعریف آرگومان موقعیتی در توابع پایتون	۲۶۹
شکل ۷-۱۲- مثالی از تعریف آرگومان کلیدواژه‌ای در توابع پایتون	۲۷۰
شکل ۷-۱۳- مثالی از تعریف آرگومان دلخواه در توابع پایتون	۲۷۱
شکل ۷-۱۴- مثالی از تفاوت بین پارامتر و آرگومان در توابع پایتون	۲۷۲

- شکل ۷-۱۵- مثالی از آرگومان دلخواه `*args` ۲۷۴
- شکل ۷-۱۶- مثالی از آرگومان دلخواه `**kwargs` ۲۷۵
- شکل ۷-۱۷- مثالی از ترکیب `*args` و `**kwargs` ۲۷۶
- شکل ۷-۱۸- مثالی از محاسبه چندین عدد با استفاده از `*args` ۲۷۷
- شکل ۷-۱۹- مثالی از نمایش یک پیام با اسمی مختلف با استفاده از `*args` ۲۷۸
- شکل ۷-۲۰- مثالی از نمایش اطلاعات یک کاربر با استفاده از `**kwargs` ۲۷۹
- شکل ۷-۲۱- مثالی از ساخت یک پیام شخصی‌سازی‌شده با `**kwargs` ۲۸۰
- شکل ۷-۲۲- مثال مربوط به توان دوم اعداد یک فهرست با استفاده از `lambda` ۲۸۶
- شکل ۷-۲۳- مثال مربوط به محاسبه مجموع دو فهرست متناظر با استفاده از `lambda` ۲۸۸
- شکل ۷-۲۴- مثال مربوط به مرتب‌سازی عناصر یک فهرست با توجه به طول عنصر با استفاده از `lambda` ۲۹۰
- شکل ۷-۲۵- مثال مربوط به الحق دو فهرست به صورت جفت با استفاده از تابع وابسته‌ی `zip` ۲۹۳
- شکل ۷-۲۶- مثال مربوط به توان دو رساندن عناصر یک فهرست با استفاده از تابع وابسته‌ی `map` ۲۹۵
- شکل ۷-۲۷- مثال مربوط به جمع اعداد دو فهرست به صورت جفت با استفاده از تابع وابسته‌ی `map` ۲۹۶
- شکل ۷-۲۸- مثال مربوط به جداسازی اعداد زوج و فرد با استفاده از تابع وابسته‌ی `filter` ۲۹۸

فصل هشتم

- شکل ۸-۱- مثالی از خطای ساختاری در پایتون ۳۰۱
- شکل ۸-۲- مثالی از خطای زمان اجرا با تقسیم عددی بر صفر ۳۰۲
- شکل ۸-۳- مثالی از خطای زمان اجرا با دادن اندیس غیرمجاز ۳۰۳

- شکل ۸-۴- مثالی از خطای منطقی به دلیل عدم رعایت اولویت‌بندی عملگرها ۳۰۴
- شکل ۸-۵- مثالی از خطای منطقی و حل آن ۳۰۵
- شکل ۸-۶- مثالی از خطای `TypeError` در پایتون ۳۰۶
- شکل ۸-۷- مثالی از خطای `ValueError` در پایتون ۳۰۷
- شکل ۸-۸- مثالی از خطای `NameError` در پایتون ۳۰۷
- شکل ۸-۹- مثالی از خطای `keyError` در پایتون ۳۰۸
- شکل ۸-۱۰- مثالی از خطای `ImportError` در پایتون ۳۰۹

فصل نهم

- شکل ۹-۱- ساختار پایه‌ای `try/except` در پایتون ۳۱۱
- شکل ۹-۲- مثالی از مدیریت خطای `ZeroDivisionError` در پایتون ۳۱۲
- شکل ۹-۳- مثالی از مدیریت چندین نوع خطا در پایتون ۳۱۳
- شکل ۹-۴- مثالی از استفاده بلوك عمومي `except` برای زمانی که نوع خطا مشخص نباشد ۳۱۴
- شکل ۹-۵- مثالی از مدیریت خطاهای با ساختار `try/except/else/finally` ۳۱۵

فصل دهم

- شکل ۱۰-۱- مثالی از دامنه محلی در پایتون ۳۱۹
- شکل ۱۰-۲- مثالی از امنه بسته در پایتون ۳۲۰
- شکل ۱۰-۳- مثالی از دامنه سراسری در پایتون ۳۲۱
- شکل ۱۰-۴- مثالی از تبدیل دامنه محلی به دامنه سراسری با کلیدواژه `global` ۳۲۲

فصل یازدهم

شکل ۱۰-۵- مثالی از تبدیل دامنه محلی به دامنه سراسری با کلیدواژه `nonlocal` ۳۲۳

شکل ۱۱-۱- ساختار کلی تابع `(__init__` در داخل کلاس‌های پایتون ۳۳۲

شکل ۱۱-۲- مثالی از تابع `__init__` در درون کلاس‌های پایتون ۳۳۳

شکل ۱۱-۳- مثالی برای درک بهتر `self` ۳۳۵

شکل ۱۱-۴- مثالی از حذف `self` در تعریف توابع وابسته ۳۳۶

شکل ۱۱-۵- مثالی از تابع `__str__` در کلاس‌های پایتون ۳۳۸

شکل ۱۱-۶- مثالی از تابع `__repr__` در کلاس‌های پایتون ۳۳۹

شکل ۱۱-۷- مثالی از ساخت کلاس بدون `__str__` و `__repr__` ۳۴۰

شکل ۱۱-۸- تفاوت بین `__str__` و `__repr__` ۳۴۱

شکل ۱۱-۹- مثالی برای تابع وابسته نمونه ۳۴۴

شکل ۱۱-۱۰- مثالی برای تابع وابسته کلاس ۳۴۵

شکل ۱۱-۱۱- مثالی برای تابع وابسته ایستا ۳۴۶

شکل ۱۱-۱۲- تعریف چند تابع وابسته‌ی مختلف در یک کلاس ۳۴۷

شکل ۱۱-۱۳- مثالی از مفهوم وراثت در پایتون ۳۴۹

شکل ۱۱-۱۴- مثالی از وراثت تک‌گانه در پایتون ۳۵۰

شکل ۱۱-۱۵- مثالی از بازنویسی کلاس والد توسط کلاس‌های فرزند ۳۵۱

شکل ۱۱-۱۶- مثال مریبوط به تابع `super` در پایتون ۳۵۳

شکل ۱۱-۱۷- مثالی از وراثت چندگانه در پایتون ۳۵۴

..... ۳۵۵	شکل ۱۱-۱۸- مثالی از وراثت چندسطحی در پایتون
..... ۳۵۸	شکل ۱۱-۱۹- مثالی از چندریختی با ارثبری
..... ۳۵۹	شکل ۱۱-۲۰- مثالی از ترکیب چند ریختی با Duck Typing

فصل دوازدهم

..... ۳۶۳	شکل ۱۲-۱- کدهای مربوط به ماژول my_module.py
..... ۳۶۴	شکل ۱۲-۲- وارد کردن ماژول my_module.py در درون پروژه main.py
..... ۳۶۶	شکل ۱۲-۳- ایجاد فایل پایتونی my_module.py به عنوان یک ماژول
..... ۳۶۷	شکل ۱۲-۴- استفاده از فایل پایتون my_module.py به عنوان ماژول در فایل پایتونی main.py
..... ۳۶۸	شکل ۱۲-۵- مثال مربوط به استفاده از __name__ در فایل اصلی
..... ۳۶۹	شکل ۱۲-۶- وارد کردن ماژول test به درون main.py
..... ۳۷۰	شکل ۱۲-۷- مثال مربوط به کدهای فایل my_module.py
..... ۳۷۰	شکل ۱۲-۸- مثال مربوط به کدهای فایل main.py

فصل سیزدهم

..... ۳۷۶	شکل ۱۳-۱- مثال مربوط به تولید عدد تصادفی بین بازه صفر تا یک با تابع وابسته random
..... ۳۷۷	شکل ۱۳-۲- مثال مربوط به تولید عدد تصادفی بین عدد ۵ تا ۱۰ با استفاده از تابع وابسته uniform
..... ۳۷۸	شکل ۱۳-۳- مثال مربوط به تولید عدد تصادفی از عدد ۵ تا ۱۰ با استفاده از تابع randint
..... ۳۷۹	شکل ۱۳-۴- مثالی از تولید عدد صحیح تصادفی از دنباله‌ای با فاصله مشخص با استفاده از تابع randrange

..... ۳۸۰ شکل ۱۳-۵- مثالی از انتخاب تصادفی یک عنصر از دنباله با استفاده از تابع وابسته choice
..... ۳۸۱ شکل ۱۳-۶- مثالی از انتخاب تصادفی ۵ عنصر از دنباله با استفاده از تابع وابسته choices
..... ۳۸۲ شکل ۱۳-۷- مثالی از چیدمان تصادفی عناصر یک فهرست با استفاده از تابع shuffle
..... ۳۸۳ شکل ۱۳-۸- مثال مربوط به نمونه‌گیری از یک فهرستبا انتخاب ۲ عنصر با استفاده از تابع وابسته sample
..... ۳۸۴ شکل ۱۳-۹- مثال مربوط به تولید عدد تصادفی با توزیع گاووسی
..... ۳۸۴ شکل ۱۳-۱۰- مثال مربوط به تولید عدد تصادفی با توزیع نمایی
..... ۳۸۵ شکل ۱۳-۱۱- مثال مربوط به تولید عدد تصادفی با توزیع بتا
..... ۳۸۶ شکل ۱۳-۱۲- تولید اعداد تصادفی تکرارپذیر با تابع وابسته seed

فصل چهاردهم

..... ۳۹۴ شکل ۱۴-۱- مثالی از نحوه محاسبه رادیکال با ماژول math
..... ۳۹۵ شکل ۱۴-۲- مثالی از محاسبه توان یک عدد با ماژول math
..... ۳۹۶ شکل ۱۴-۳- مثالی از محاسبه فاکتوریل یک عدد با ماژول math
..... ۳۹۷ شکل ۱۴-۴- مثالی از محاسبه مجموع اعداد یک فهرست با استفاده از تابع وابسته fsum
..... ۳۹۸ شکل ۱۴-۵- مثال‌های مربوط به سینوس، کسینوس و تانژانت با استفاده از ماژول math
..... ۳۹۹ شکل ۱۴-۶- مثال‌های مربوط به محاسبه سینوس معکوس، کسینوس معکوس و تانژانت معکوس توسط ماژول math
..... ۴۰۰ شکل ۱۴-۷- مثال مربوط به تبدیل رادیان به درجه با استفاده از ماژول math
..... ۴۰۱ شکل ۱۴-۸- مثال مربوط به تبدیل درجه به رادیان با استفاده از ماژول math

- شکل ۱۴-۹- مثال مربوط به محاسبه توابع نمایی با استفاده از ماژول `math`
 ۴۰۲.....
- شکل ۱۴-۱۰- مثال مربوط به محاسبه لگاریتم یک عدد با اعمال مبنای دلخواه با استفاده از ماژول `math`
 ۴۰۳.....
- شکل ۱۴-۱۱- مثال مربوط به محاسبه لگاریتم یک عدد با مبنای ۱۰ با استفاده از ماژول `math` ۴۰۴.....
- شکل ۱۴-۱۲- مثال مربوط به محاسبه لگاریتم یک عدد با مبنای ۲ با استفاده از ماژول `math` ۴۰۵.....
- شکل ۱۴-۱۳- مثال مربوط به گرد کردن عددی به سمت عدد بزرگتر از خودش توسط ماژول `math` ۴۰۶.....
- شکل ۱۴-۱۴- مثال مربوط به گرد کردن عددی به سمت عدد کوچکتر از خودش توسط ماژول `math` ۴۰۷...
 شکل ۱۴-۱۵- مثال مربوط به حذف قسمت اعشاری یک عدد با استفاده از ماژول `math` ۴۰۸.....
- شکل ۱۴-۱۶- مثالی از جداسازی قسمت اعشاری و صحیح یک عدد توسط ماژول `math` ۴۰۹.....
- شکل ۱۴-۱۷- عدد پی در ماژول `math` ۴۱۰.....
- شکل ۱۴-۱۸- عدد نپر در ماژول `math` ۴۱۱.....
- شکل ۱۴-۱۹- عدد تاو در ماژول `math` ۴۱۱.....
- شکل ۱۴-۲۰- مقدار بینهایت در ماژول `math` ۴۱۲.....

فصل پانزدهم

- شکل ۱۵-۱- مالی از گرفتن تاریخ و زمان فعلی سیستم با استفاده از ماژول `datetime` ۴۲۱.....
- شکل ۱۵-۲- مثالی از گرفتن تاریخ و زمان فعلی سیستم بدون تنظیمات منطقه زمانی با استفاده از ماژول `datetime` ۴۲۲.....
- شکل ۱۵-۳- مثالی از تبدیل یک رشته تاریخ/زمان به شیء `datetime` ۴۲۳.....
- شکل ۱۵-۴- مثالی از تبدیل یک شیء `datetime` به رشته تاریخ/زمان ۴۲۴.....

- شکل ۱۵-۵- مثالی از برگرداندن تاریخ و زمان معادل یک زمان یونیکس با استفاده از مازول **datetime** ۴۲۵
- شکل ۱۵-۶- مثالی از ترکیب یک تاریخ و زمان و ایجاد یک شیء **datetime** ۴۲۵
- شکل ۱۵-۷- مثالی از تغییر دادن قسمتی از یک شیء **datetime** ۴۲۶
- شکل ۱۵-۸- مثالی از برگرداندن زمان یونیکس معادل یک شیء **datetime** ۴۲۷

فصل شانزدهم

- شکل ۱۶-۱- مثال مربوط به ایجاد ماتریس صفر 3×3 با استفاده از مازول نامپای ۴۳۴
- شکل ۱۶-۲- مثال مربوط به ایجاد ماتریس 3×3 با درایه‌های یک با استفاده از مازول نامپای ۴۳۵
- شکل ۱۶-۳- مثال مربوط به ایجاد ماتریس قطری 3×3 با استفاده از مازول نامپای ۴۳۶
- شکل ۱۶-۴- مثال مربوط به ایجاد آرایه‌های چندبعدی در نامپای ۴۳۷
- شکل ۱۶-۵- مثالی از عملیات جمع بین ماتریس‌ها در مازول نامپای ۴۳۸
- شکل ۱۶-۶- مثالی از ضرب ماتریسی در مازول نامپای ۴۳۹
- شکل ۱۶-۷- مثالی از جابجایی عناصر قطری یک ماتریس در مازول نامپای ۴۴۰
- شکل ۱۶-۸- مثال مربوط به محاسبه معکوس ماتریس در مازول نامپای ۴۴۱
- شکل ۱۶-۹- مثال مربوط به محاسبه دترمینال ماتریس در مازول نامپای ۴۴۲
- شکل ۱۶-۱۰- مثالی از دسترسی به عنصر خاصی از ماتریس ۴۴۳
- شکل ۱۶-۱۱- مثالی از مثالی از دسترسی به سطر یا ستون مورد نظر از ماتریس ۴۴۴
- شکل ۱۶-۱۲- مثال مربوط به تغییر شکل ماتریس ۴۴۵
- شکل ۱۶-۱۳- مثال مربوط به ایجاد آرایه یک بعدی و دو بعدی با استفاده از مازول نامپای ۴۴۶
- شکل ۱۶-۱۴- مثال مربوط به ایجاد آرایه‌های خاص ۴۴۸

شکل ۱۶-۱۵- مثالی از به دست آوردن اندازه، شکل و بعد یک آرایه با استفاده از ماثول نامپای ۴۴۹
شکل ۱۶-۱۶- مثال مربوط به عملیات جمع و تفریق، ضرب و تقسیم و توان دوم برای دو آرایه ۴۵۰
شکل ۱۶-۱۷- مثالی از اعمال برخی اعمال ریاضی روی درایه‌های یک آرایه ۴۵۲

فصل هفدهم

شکل ۱۷-۱- مثالی از ساده‌سازی عبارات جبری با استفاده از ماثول SymPy ۴۶۰
شکل ۱۷-۲- مثالی از گسترش عبارات جبری با استفاده از ماثول SymPy ۴۶۱
شکل ۱۷-۳- مثالی از محاسبه حد زمانی که X به سمت یک میل می‌کند ۴۶۳
شکل ۱۷-۴- مثالی از محاسبه حد زمانی که X به سمت بی‌نهایت میل می‌کند ۴۶۴
شکل ۱۷-۵- مثالی از محاسبه حد زمانی که X به سمت صفر مثبت میل می‌کند ۴۶۵
شکل ۱۷-۶- مثالی از محاسبه حد زمانی که X به سمت صفر منفی میل می‌کند ۴۶۶
شکل ۱۷-۷- مثالی از محاسبه حد توابع مثلثاتی زمانی که X به سمت صفر میل می‌کند ۴۶۷
شکل ۱۷-۸- مثالی از محاسبه حد برای نقاط ناپیوسته ۴۶۸
شکل ۱۷-۹- مثالی از محاسبه مشتق اول توسط ماثول SymPy ۴۶۹
شکل ۱۷-۱۰- مثالی از محاسبه مشتق دوم توسط ماثول SymPy ۴۷۰
شکل ۱۷-۱۱- مثالی از محاسبه مشتق توابع چند متغیره توسط ماثول SymPy ۴۷۱
شکل ۱۷-۱۲- مثالی از محاسبه مشتق توابع مثلثاتی توسط ماثول SymPy ۴۷۲
شکل ۱۷-۱۳- مثالی از محاسبه مشتق توابع مرکب توسط ماثول SymPy ۴۷۳
شکل ۱۷-۱۴- مثالی از محاسبه انتگرال نامعین با استفاده از ماثول SymPy ۴۷۵
شکل ۱۷-۱۵- مثالی از محاسبه انتگرال معین با استفاده از ماثول SymPy ۴۷۶

شکل ۱۷-۱۶- مثالی از محاسبه انتگرال توابع مثلثاتی با استفاده از ماژول SymPy	۴۷۷
شکل ۱۷-۱۷- مثالی از محاسبه انتگرال توابع چند متغیره با استفاده از ماژول SymPy	۴۷۸
شکل ۱۷-۱۸- مثالی از محاسبه انتگرال عددی با استفاده از ماژول SymPy	۴۷۹
شکل ۱۷-۱۹- مثالی از محاسبه انتگرال گیری از توابع پیچیده با استفاده از ماژول SymPy	۴۸۰
شکل ۱۷-۲۰- ساختار کلی استفاده از تابع solve برای حل معادلات جبری	۴۸۲
شکل ۱۷-۲۱- مثالی از حل معادله خطی با استفاده از ماژول SymPy	۴۸۲
شکل ۱۷-۲۲- مثالی از حل معادله درجه دوم با استفاده از ماژول SymPy	۴۸۳
شکل ۱۷-۲۳- مثالی از حل سیستم معادلات با استفاده از ماژول SymPy	۴۸۴
شکل ۱۷-۲۴- مثالی از حل معادلات غیرخطی با استفاده از ماژول SymPy	۴۸۵
شکل ۱۷-۲۵- مثالی از حل معادلات با solveset در ماژول SymPy	۴۸۶
شکل ۱۷-۲۶- مثالی از حل معادله دیفرانسیلی ساده با استفاده از ماژول SymPy	۴۸۸
شکل ۱۷-۲۷- مثالی از حل معادله دیفرانسیلی مرتبه دوم با استفاده از ماژول SymPy	۴۸۹
شکل ۱۷-۲۸- مثالی از حل معادلات دیفرانسیلی غیرخطی با استفاده از ماژول SymPy	۴۹۰
شکل ۱۷-۲۹- مثال مربوط به ساده‌سازی عبارت ریاضی	۴۹۱
شکل ۱۷-۳۰- مثال مربوط به حل معادله درجه دوم	۴۹۲
شکل ۱۷-۳۱- مثال مربوط به محاسبه مشتق یک تابع	۴۹۳
شکل ۱۷-۳۲- مثال مربوط به محاسبه انتگرال یک تابع	۴۹۴
شکل ۱۷-۳۳- مثال مربوط به پیاده‌سازی سری تیلور در ماژول SymPy	۴۹۵
شکل ۱۷-۳۴- مثال مربوط به پیاده‌سازی مساحت دایره به صورت تحلیلی در ماژول SymPy	۴۹۶

فصل هجدهم

- شکل ۱۸-۱- بارگذاری داده‌ها از فایل CSV ۵۰۳
شکل ۱۸-۲- پروژه مربوط به محاسبه میانگین ۴۰ دانشجو با استفاده از pandas ۵۰۸

فصل نوزدهم

- شکل ۱۹-۱- مثال مربوط به ترسیم نمودار خطی با استفاده از matplotlib ۵۱۲
شکل ۱۹-۲- خروجی مربوط به نمودار خطی (شکل ۱-۱۹) ۵۱۳
شکل ۱۹-۳- مثال مربوط به ترسیم نمودار میله‌ای با استفاده از matplotlib ۵۱۴
شکل ۱۹-۴- خروجی مربوط به نمودار میله‌ای (شکل ۳-۱۹) ۵۱۴
شکل ۱۹-۵- مثال مربوط به ترسیم نمودار دایره‌ای با استفاده از matplotlib ۵۱۵
شکل ۱۹-۶- خروجی مربوط به نمودار دایره‌ای (شکل ۵-۱۹) ۵۱۵
شکل ۱۹-۷- مثال مربوط به ترسیم نمودار پراکندگی با استفاده از matplotlib ۵۱۶
شکل ۱۹-۸- خروجی مربوط به نمودار پراکندگی (شکل ۷-۱۹) ۵۱۷
شکل ۱۹-۹- مثال مربوط به ترسیم نمودار هیستوگرام با استفاده از matplotlib ۵۱۷
شکل ۱۹-۱۰- خروجی مربوط به نمودار هیستوگرام (شکل ۹-۱۹) ۵۱۸
شکل ۱۹-۱۱- مثالی از ترسیم چند نمودار در یک صفحه در matplotlib ۵۱۹
شکل ۱۹-۱۲- خروجی مربوط به ترسیم چند نمودار (شکل ۱۱-۱۹) ۵۲۰
شکل ۱۹-۱۳- مثال مربوط به ترسیم نمودار سه‌بعدی در matplotlib ۵۲۱
شکل ۱۹-۱۴- خروجی مربوط به ترسیم نمودار سه‌بعدی (شکل ۱۳-۱۹) ۵۲۲
شکل ۱۹-۱۵- مثال مربوط به ترسیم نمودارهای log-log با استفاده از تابع loglog ۵۲۳

۵۲۴-۱۶- خروجی مربوط به ترسیم نمودارهای log-log با استفاده از تابع `loglog` (شکل ۱۹-۱۹) شکل ۱۹-۱۹

۵۲۵-۱۷- مثال مربوط به ترسیم نمودارهای log-log با تنظیم دستی مقیاس محورها شکل ۱۹-۱۸

۵۲۶-۱۸- خروجی مربوط به ترسیم نمودارهای log-log با تنظیم دستی مقیاس محورها (شکل ۱۸-۱۹) شکل ۱۹-۱۹

۵۲۷-۱۹- مثال مربوط به ترسیم نمودارهای log-log با ترکیب چند نمودار در log-log (شکل ۱۹-۲۰) شکل ۱۹-۲۰

۵۲۸-۲۰- خروجی مربوط به ترسیم نمودارهای log-log با ترکیب چند نمودار در log-log (شکل ۱۹-۲۱) شکل ۱۹-۲۱

۵۳۰-۲۱- مثال مربوط به ترسیم نمودارهای Semi-log با استفاده از تابع `x` (شکل ۱۹-۲۲) شکل ۱۹-۲۲

۵۳۱-۲۲- خروجی مربوط به ترسیم نمودارهای Semi-log با استفاده از تابع `x` (شکل ۱۹-۲۳) شکل ۱۹-۲۳

۵۳۲-۲۳- مثال مربوط به ترسیم نمودارهای Semi-log با استفاده از تابع `y` (شکل ۱۹-۲۴) شکل ۱۹-۲۴

۵۳۳-۲۴- خروجی مربوط به ترسیم نمودارهای Semi-log با استفاده از تابع `y` (شکل ۱۹-۲۵) شکل ۱۹-۲۵

۵۳۴-۲۵- مثال مربوط به ترسیم نمودارهای Semi-log با ترکیب چند نمودار در log-log (شکل ۱۹-۲۶) شکل ۱۹-۲۶

۵۳۵-۲۶- خروجی مربوط به ترسیم نمودارهای Semi-log با ترکیب چند نمودار در log-log (شکل ۱۹-۲۷) شکل ۱۹-۲۷

۵۳۷-۲۷- مثال مربوط به انواع برچسب‌گذاری در نمودارهای `matplotlib` شکل ۱۹-۲۸

۵۳۷-۲۸- خروجی مربوط به انواع برچسب‌گذاری در `matplotlib` (شکل ۱۹-۲۹) شکل ۱۹-۲۹

۵۳۸-۲۹- مثال مربوط به تنظیم محدوده محورها در `matplotlib` شکل ۱۹-۳۰

۵۳۹-۳۰- خروجی مربوط به تنظیم محدوده محورها (شکل ۱۹-۲۹) شکل ۱۹-۳۱

..... شکل ۱۹ -۳۲ - خروجی مربوط به افزودن شبکه به محورها (شکل ۱۹-۳۱)	۵۴۰
..... شکل ۱۹ -۳۳ - مثال مربوط به برچسب‌گذاری دلخواه برای محورها	۵۴۱
..... شکل ۱۹ -۳۴ - خروجی مربوط به برچسب‌گذاری دلخواه برای محورها (شکل ۱۹-۳۳)	۵۴۲
..... شکل ۱۹ -۳۵ - مثال مربوط به چرخش برچسب‌های محورها	۵۴۳
..... شکل ۱۹ -۳۶ - خروجی مربوط به چرخش برچسب‌های محورها (شکل ۱۹-۳۵)	۵۴۳
..... شکل ۱۹ -۳۷ - مثال مربوط به افزودن خطوط به محورها	۵۴۴
..... شکل ۱۹ -۳۸ - خروجی مربوط به افزودن خطوط به محورها (شکل ۱۹-۳۷)	۵۴۵
..... شکل ۱۹ -۳۹ - مثال مربوط به ترسیم دو نمودار در یک ردیف و دو ستون	۵۴۶
..... شکل ۱۹ -۴۰ - خروجی مربوط به ترسیم دو نمودار در یک ردیف و دو ستون (شکل ۱۹-۳۹)	۵۴۷
..... شکل ۱۹ -۴۱ - مثال مربوط به ترسیم چهار نمودار در دو ردیف و دو ستون	۵۴۸
..... شکل ۱۹ -۴۲ - خروجی مربوط به ترسیم چهار نمودار در دو ردیف و دو ستون (شکل ۱۹-۴۱)	۵۴۹
..... شکل ۱۹ -۴۳ - مثال مربوط به روش دوم برای ترسیم چند نمودار در یک صفحه	۵۵۰
..... شکل ۱۹ -۴۴ - خروجی مربوط به روش دوم برای ترسیم چند نمودار در یک صفحه (شکل ۱۹-۴۳)	۵۵۱
..... شکل ۱۹ -۴۵ - مثال مربوط به نمودار هیستوگرام	۵۵۳
..... شکل ۱۹ -۴۶ - خروجی مربوط به نمودار هیستوگرام (شکل ۱۹-۴۵)	۵۵۳
..... شکل ۱۹ -۴۷ - مثال مربوط به نمودار میله‌ای	۵۵۵
..... شکل ۱۹ -۴۸ - خروجی مربوط به نمودار میله‌ای (شکل ۱۹-۴۷)	۵۵۵
..... شکل ۱۹ -۴۹ - مثال مربوط به ترکیب نمودارهای هیستوگرام و میله‌ای	۵۵۷
..... شکل ۱۹ -۵۰ - خروجی مربوط به ترکیب نمودارهای هیستوگرام و میله‌ای (شکل ۱۹-۴۹)	۵۵۸

فهرست جداول

فصل اول

جدول ۱ - ۱- مقایسه زبان‌های برنامه‌نویسی پایتون و سی	۵۴
جدول ۱ - ۲- مقایسه زبان‌های برنامه‌نویسی پایتون و سی‌پلاس‌پلاس	۵۵
جدول ۱ - ۳- مقایسه زبان‌های برنامه‌نویسی پایتون و سی‌شارپ	۵۷
جدول ۱ - ۴- مقایسه زبان‌های برنامه‌نویسی پایتون و جاوا	۵۸
جدول ۱ - ۵- مقایسه زبان‌های برنامه‌نویسی پایتون و متلب	۶۰

فصل دوم

جدول ۲ - ۱- بررسی ویژگی‌های زبان برنامه‌نویسی پایتون	۶۴
جدول ۲ - ۲- بررسی کاربردهای زبان برنامه‌نویسی پایتون	۶۶

فصل سوم

جدول ۳ - ۱- پارامترهای مربوط به تابع print	۸۷
جدول ۳ - ۲- عملگرهای ریاضی در پایتون به همراه مثال	۹۳
جدول ۳ - ۳- انواع عملگرهای مقایسه‌ای در پایتون به همراه مثال	۹۴
جدول ۳ - ۴- انواع عملگرهای تخصیص در پایتون به همراه مثال	۹۶
جدول ۳ - ۵- انواع عملگرهای منطقی در پایتون به همراه مثال	۹۷
جدول ۳ - ۶- مقایسه‌ی دو شرط با عملگر منطقی and	۹۸

جدول ۳-۷- مقایسه‌ی دو شرط با عمل‌گر منطقی *Or* ۹۹

جدول ۳-۸- اولویت‌بندی عمل‌گرها در پایتون ۱۰۰

فصل چهارم

جدول ۴-۱- انواع داده‌ها در پایتون ۱۰۴

جدول ۴-۲- جدول مربوط به توابع وابسته موجود در نوع داده‌ی رشته‌ای ۱۱۱

جدول ۴-۳- جدول مربوط به توابع وابسته موجود در نوع داده‌ی فهرست ۱۳۴

جدول ۴-۴- جدول مربوط به توابع وابسته موجود در نوع داده‌ی دیکشنری ۱۵۶

جدول ۴-۵- جدول مربوط به توابع وابسته موجود در نوع داده‌ی تاپل ۱۷۸

جدول ۴-۶- جدول مربوط به توابع وابسته موجود در نوع داده‌ی مجموعه ۱۹۱

فصل هفتم

جدول ۷-۱- جدول مربوط به توابع درونی موجود در پایتون ۲۸۱

فصل سیزدهم

جدول ۱۳-۱- جدول مربوط به تمامی توابع وابسته‌ی موجود در ماژول *random* ۳۷۵

فصل چهاردهم

جدول ۱۴-۱- جدول مربوط به تمامی توابع وابسته‌ی موجود در ماژول *Math* ۳۹۲

فصل پانزدهم

جدول ۱۵ - ۱ - جدول مربوط به تمامی توابع وابسته‌ی موجود در ماژول `Datetime` ۴۲۰

فصل هجدهم

جدول ۱۸ - ۱ - جدول مربوط به مقایسه `pandas` با سایر ماژول‌ها ۵۰۱

فصل نوزدهم

جدول ۱۹ - ۱ - جدول مربوط به تفاوت‌های نمودار‌های هیستوگرام و میله‌ای ۵۵۶

فصل اول

آشنایی با برنامهنویسی

۱) آشنایی با برنامه‌نویسی و زبان‌های برنامه‌نویسی

برنامه‌نویسی^۱، هنری است که در آن، ایده‌های انتزاعی و مفاهیم ریاضی به دستورالعمل‌هایی تبدیل می‌شوند که می‌توانند کامپیوترها را برای انجام وظایف گوناگون به کار گیرند. از زمان پیدایش کامپیوترها تا به امروز، برنامه‌نویسی به یکی از مؤلفه‌های اساسی در دنیای فناوری تبدیل شده و نقش مهمی در توسعه ابزارها، اپلیکیشن‌ها و سیستم‌هایی ایفا کرده است که زندگی روزمره‌ی ما را دگرگون ساخته‌اند.

در مفهوم کلی، برنامه‌نویسی به فرآیند نوشتمن کد یا دستوراتی گفته می‌شود که کامپیوتر می‌تواند آن‌ها را بخواند و اجرا کند. این دستورات، به صورت گام‌به‌گام به کامپیوتر گفته می‌شود تا کار خاصی را انجام دهد؛ برای مثال، محاسبه یک عدد، نمایش اطلاعات، کنترل حرکت یک ربات یا ایجاد محیط‌های گرافیکی و تعاملی. برنامه‌نویسی به کمک زبان‌های خاصی انجام می‌شود که این زبان‌ها رابطی بین تفکر انسانی و منطق دیجیتال ماشین‌ها هستند. زبان‌های برنامه‌نویسی^۲ مختلفی با ویژگی‌ها و قابلیت‌های گوناگون وجود دارند؛ برخی ساده و ابتدایی و برخی دیگر پیچیده و قدرتمندند.

۱-۱) اهمیت و نقش برنامه‌نویسی

برنامه‌نویسی امروز تنها محدود به کامپیوترها نیست؛ بلکه دامنه‌ی آن به عرصه‌های وسیعی از زندگی مدرن کشیده شده است. صنایع مختلف، از سلامت و پزشکی گرفته تا تجارت و سرگرمی، همگی به نوعی از برنامه‌نویسی بهره می‌گیرند. مهارت‌های برنامه‌نویسی به ما امکان می‌دهند تا نه تنها از فناوری‌های موجود بهره‌مند شویم، بلکه در ایجاد و بهبود آن‌ها نیز نقش داشته باشیم. همچنین، یادگیری برنامه‌نویسی تفکر تحلیلی، حل مسئله و خلاقیت را در افراد تقویت می‌کند و آنان را برای مواجهه با چالش‌های پیچیده آماده می‌سازد.

¹ Programming

² Programming Languages

۱-۲) گام‌های اولیه در یادگیری برنامه‌نویسی

برای ورود به دنیای برنامه‌نویسی، آشنایی با مفاهیم پایه‌ای ضروری است. این مفاهیم شامل اصول الگوریتم‌نویسی، آشنایی با ساختارهای داده، و نحوه حل مسائل به روش‌های مختلف می‌باشد. همچنین، انتخاب یک زبان برنامه‌نویسی مناسب برای شروع، یکی از گام‌های مهم محسوب می‌شود. زبان‌هایی مانند پایتون، جاوا اسکریپت و جاوا برای مبتدیان مناسب‌اند، زیرا ساختار آن‌ها ساده و یادگیری آن‌ها آسان است. هر زبان برنامه‌نویسی، کاربردها و ویژگی‌های خاص خود را دارد که براساس نوع پروژه و اهداف برنامه‌نویس، می‌تواند مورد استفاده قرار گیرد.

۱-۳) خلاقیت در برنامه‌نویسی

برنامه‌نویسی تنها به نوشتمن کد و حل مسائل محدود نمی‌شود؛ بلکه یکی از جنبه‌های جذاب آن، "خلاقیت" است. برنامه‌نویسان به کمک کد، ایده‌های خود را به واقعیت تبدیل می‌کنند و توانایی ایجاد ابزارها و برنامه‌های جدید را به دست می‌آورند. از توسعه‌ی بازی‌های کامپیوتری گرفته تا طراحی هوش مصنوعی، همه و همه باعث می‌شود تا فرصتی برای کشف و خلق ایده‌ها با برنامه‌نویسی فراهم شود.

در نهایت، "برنامه‌نویسی" سفری جذاب به دنیای دیجیتال است که با هر گام، به درک عمیق‌تر از فناوری و ایجاد تغییرات مثبت در جهان اطرافمان منجر می‌شود. این مسیر، نه تنها درهای جدیدی از دانش و مهارت را به روی ما می‌گشاید، بلکه آینده‌ای پر از امکان و خلاقیت را نوید می‌دهد.

۱-۴) تعریف برنامه‌نویسی

همان‌طور که در دنیای واقعی زبان انگلیسی به زبان بین‌المللی و مشترک میان افراد مختلف در جهان تبدیل شده است، در جهان فناوری و کامپیوتر هم برنامه‌نویسی به عنوان زبان برقراری ارتباط بین ماشین و انسان درنظر گرفته

شده است. در حال حاضر برنامه‌نویسی به یکی از پرطرفدارترین مشاغل دنیا تبدیل شده است. در عصر دیجیتال، بحث در مورد برنامه‌نویسی در همه جا گسترش یافته است. اما برنامه‌نویسی دقیقاً به چه معناست؟

به زبان ساده، برنامه‌نویسی یعنی نوشتن دستورات منطقی برای کامپیوترها و ماشین‌ها. به عبارت دیگر، برنامه‌نویسی فرآیند ایجاد دستورالعمل‌هایی برای کامپیوتر است تا وظایف خاصی را انجام دهد. این دستورالعمل‌ها به زبان‌های برنامه‌نویسی نوشته می‌شوند که مجموعه‌ای از قوانین و ساختارهای خاص دارند که کامپیوتر می‌تواند آن‌ها را درک و اجرا کند. این دستورهای منطقی را با استفاده از زبان‌های برنامه‌نویسی که توسط کامپیوترها قابل فهم هستند، می‌نویسن. سپس کامپیوتر با تحلیل و پردازش دستورها، آن‌ها را اجرا می‌کند. برنامه‌نویسی به ما این امکان را می‌دهد که با هر سیستم یا ماشین الکترونیکی ارتباط برقرار کنیم و آنها را برنامه‌ریزی کنیم.

برای مثال فرض کنید شخصی با سطح هوشمندی کم می‌خواهد یک اسباب‌بازی لگو را بسازد. این شخص دفترچه راهنمای ساخت لگو را در اختیار ندارد و تنها می‌تواند بر اساس دستورات شما ساخت لگو را انجام دهد. باید به یاد داشت که این شخص فاقد هوشمندی است و در صورتی که دستورالعمل‌های دقیق و مشخصی را در خصوص نحوه ساخت لگو دریافت نکند، به احتمال زیاد اشتباهات بسیاری را مرتکب خواهد شد. اگر نحوه تفکر این شخص مثل یک کامپیوتر باشد، آنوقت حتی اگر دستورالعمل مربوط به تنها یک قطعه لگو و نحوه قرار دادن آن در محل صحیح به طور مشخص تعیین نشود، کل فرآیند ساخت اسباب‌بازی لگو با مشکل مواجه خواهد شد. در واقع، دستور دادن به این شخص فاقد هوشمندی، بسیار شبیه به نحوه انجام برنامه‌نویسی است. با این تفاوت که در واقعیت به جای یک شخص فاقد هوشمندی، با یک کامپیوتر فاقد هوشمندی سرو کار دارد.

جالب است بدانید که امروزه تمام سیستم‌هایی که کوچکترین اثری از هوشمندسازی در آن‌ها دیده می‌شود، توسط زبان‌های برنامه‌نویسی، ساخته شده‌اند. تک تک کارهایی که قادر هستید با استفاده از موبایل‌تان انجام دهید، مانند فرستادن پیام، تماس، پخش ویدیو، پخش موسیقی و... با برنامه‌نویسی ایجاد شدند. امروزه اشیا و وسائل

خانه نیز قابلیت برنامه‌نویسی دارند و نسل جدید لباسشویی، یخچال و... همگی با استفاده از این حرفه، هوشمندسازی می‌شوند.

۱-۵) تاریخچه برنامه‌نویسی

برنامه‌نویسی اولین بار در سال ۱۸۸۳ توسط یک ریاضی‌دان و نویسنده انگلیسی به نام آیدا لاولیس^۳، زمانی که بر روی پروژه موتور تحلیلی دانشمند معروف چارلز ببیج^۴ کار می‌کرد، کشف شد. او متوجه شد این کامپیوتر ابتدایی می‌تواند کارهای پیچیده‌تری از محاسبات ساده ریاضی را انجام دهد، از این رو او عنوان اولین برنامه‌نویس جهان را به نام خود ثبت کرد.

در دهه ۱۹۴۰ با ابداع ماشین تورینگ توسط یک ریاضیدان بریتانیایی، انقلابی بزرگ در حوزه برنامه‌نویسی رخداد تا مقدمه ساز ایجاد زبان‌های برنامه‌نویسی شود. در دهه ۱۹۵۰ زبان اسembly^۵ که اولین زبان برنامه‌نویسی سطح پایین بود ظاهر شد. درست در طی یک دهه پس از آن، رایانه‌ها به عنوان ابزارهای اصلی برنامه‌نویسی وارد عرصه شدند و پس از آن زبان فورترن^۶ که با هدف حل محاسبات علمی و ساده‌سازی عملیات ریاضی طراحی شده بود، ظهر کرد و به عنوان اولین زبان برنامه‌نویسی علمی شناخته شد. از سال ۱۹۷۰ زبان‌های برنامه‌نویسی مدرن مانند پایتون، جاوا اسکریپت، روبي و ... ساخته شدند و تا امروز که زبان‌های متعدد دیگری به وجود آمده‌اند.

۱-۶) زبان‌های برنامه‌نویسی

دنیای زبان‌های برنامه‌نویسی؛ دروازه‌ای به سوی خلق ایده‌ها؛

³ Ada Lovelace

⁴ Charles Babbage

⁵ Assembly Language

⁶ Fortran

برنامه‌نویسی هنر ترجمه ایده‌ها به زبان کامپیوتر است. با استفاده از زبان‌های برنامه‌نویسی، دستوراتی به کامپیوتر می‌دهید تا وظایف مختلفی را انجام دهد، مانند راهاندازی یک وبسایت ساده یا طراحی مدل‌های هوش مصنوعی پیچیده. زبان‌های برنامه‌نویسی حکم الفبا را در دنیای دیجیتال دارند. هر کدام از این زبان‌ها با قواعد و دستورالعمل‌های خاص خود، دنیایی از امکانات را پیش روی ما می‌گشایند. زبان‌های برنامه‌نویسی مترجم و واسط بین زبان ماشین و زبان انسان می‌باشد. یادگیری زبان‌های برنامه‌نویسی نسبت به یادگیری کدهای صفر و یک ماشین بسیار ساده‌تر می‌باشد. می‌توان زبان‌های برنامه‌نویسی را چیزی بین زبان ماشین و زبان محاوره انسان‌ها تصور کرد. زبان‌های برنامه‌نویسی بسته به کاربرد، نوع اجرا و پیچیدگی به انواع مختلفی تقسیم می‌شوند. یکی از این دسته‌بندی‌ها، رتبه‌بندی زبان‌های برنامه‌نویسی براساس میزان نزدیک بودن به زبان انسان (زبان انگلیسی) است. بر همین اساس هر زبان برنامه‌نویسی در یکی از دو گروه سطح بالا و سطح پایین قرار می‌گیرد.

۱-۶-۱) زبان‌های برنامه‌نویسی سطح پایین

زبان‌های سطح پایین⁷، با زبان ماشین (زبان ذاتی کامپیوتر) سخن می‌گویند. گویی این زبان‌ها، مستقیماً با سختافزار کامپیوتر در تعامل هستند. به عبارت دیگر زبان‌های برنامه‌نویسی سطح پایین بیشتر به زبان ماشین یا کامپیوتر نزدیک است تا به زبان انسان‌ها. این زبان‌ها به ما قدرت کنترل کامل بر منابع سیستم را می‌دهند. برنامه‌های نوشته شده با زبان‌های سطح پایین، به دلیل همگامی مستقیم با سختافزار، بسیار سریع و کارآمد هستند. از معروف‌ترین زبان‌های سطح پایین می‌توان به زبان‌هایی مانند اسمنبلی، سی و سی‌پلاس‌پلاس اشاره کرد.

⁷ Low-level programming language

۱-۶-۲) زبان‌های برنامه‌نویسی سطح بالا

زبان‌های سطح بالا^۸ با زبانی نزدیک به زبان انسان، دستورات را به کامپیوتر ابلاغ می‌کنند. گویی با این زبان‌ها، از پیچیدگی‌های ریز و جزئیات فنی فاصله می‌گیرید. این زبان‌ها با استفاده از دستورالعمل‌ها و کلمات کلیدی ساده، برنامه‌نویسی را آسان‌تر و قابل فهم‌تر می‌کنند. از طرفی دیگر، با استفاده از این زبان‌ها، می‌توانید بر منطق برنامه و کارکرد اصلی آن تمرکز کنید و درگیر پیچیدگی‌های فنی سطح پایین نشوید. برخی از محبوب‌ترین زبان‌های سطح بالا:

- ✓ پایتون: زبان محبوب مبتدیان به دلیل سادگی و خوانایی، کاربرد گسترده در هوش مصنوعی و یادگیری ماشین.
- ✓ جاوا: زبان قدرتمند و همه‌کاره برای وب‌اپلیکیشن‌ها، برنامه‌های اندرویدی و موارد دیگر.
- ✓ جاوا اسکریپت: زبان ضروری برای خلق وب‌سایت‌های تعاملی و پویا.
- ✓ سی پلاس پلاس: زبان سریع و کارآمد برای برنامه‌های سیستمی، بازی‌ها و شبیه‌سازی‌ها.
- ✓ سی شارپ: زبان قدرتمند برای برنامه‌نویسی تحت ویندوز و وب‌اپلیکیشن‌ها.
- ✓ پی اچ پی: زبان محبوب برای برنامه‌نویسی سمت سرور و وب‌سایت‌های پویا.
- ✓ متلب: زبان قدرتمند برای استفاده در کاربردهای مهندسی و محاسبات پیچیده ریاضی.

۱-۶-۳) آشنایی با زبان برنامه‌نویسی پایتون^۹

زبان برنامه‌نویسی پایتون در سال ۱۹۹۱ توسط شخصی به نام خیدو فان روسم^{۱۰} توسعه داده شد. پایتون یک زبان برنامه‌نویسی شی گرا^{۱۱} و سطح بالا است؛ سطح بالا بودن پایتون یکی از مهم‌ترین دلایل محبوبیت این زبان

⁸ High-level programming language

⁹ Python Programming Language

¹⁰ Guido van Rossum

¹¹ Object Oriented Programming

است، ساختار زبان پایتون تشکیل شده از کلمات انگلیسی و اعداد ریاضی است که یادگیری و استفاده از آن را بسیار ساده کرده است.

زبان برنامهنویسی پایتون، زبانی با یادگیری آسان محسوب می‌شود و از همین رو اولین زبان برنامهنویسی برای تازه‌کارها، زبان برنامهنویسی پایتون می‌باشد؛ زیرا پایتون به عنوان یک زبان "همه‌منظور" ساخته و توسعه داده شده و محدود به توسعه نوع خاصی از نرم‌افزارها نیست. به بیان دیگر، می‌توان از آن برای هر کاری، از تحلیل داده گرفته تا ساخت بازی‌های کامپیوترا استفاده کرد.

۱-۶-۴) آشنایی با زبان برنامهنویسی سی

در دنیای امروز از تلفن‌های هوشمند گرفته تا نرم‌افزارهای مختلف که با کمک زبان‌های برنامهنویسی ساخته شدند برای انجام کارها مورد استفاده قرار می‌گیرند. زبان‌های برنامهنویسی مختلفی وجود دارد که برای کدنویسی مورد استفاده قرار می‌گیرند اما مدعی‌ترین و تاثیرگذارترین زبان در بین این‌ها، زبان برنامهنویسی سی^{۱۲} می‌باشد. زبان سی یکی از پرکاربردترین زبان‌های برنامهنویسی در دنیا می‌باشد که از اهمیت بالایی در علوم کامپیوترا برخوردار است چراکه زبان سی مادر همه زبان‌های برنامهنویسی می‌باشد و از طرفی ریشه و اساس زبان‌هایی مثل سی‌پلاس‌پلاس، سی‌شارپ، جاوا، جاوا اسکریپت، PHP، پایتون و ... به حساب می‌آید. با اینکه حدود پنجاه سال از زمان به وجود آمدن این زبان می‌گذرد اما هم‌چنان زبان برنامهنویسی سی در بین ده زبان محبوب و پرطرفدار در دنیا قرار دارد. دنیس ریچی^{۱۳} در بین سال‌های ۱۹۶۹ تا ۱۹۷۳، زبان برنامهنویسی سی را ساخت.

زبان برنامهنویسی سی انتخابی مناسب برای برنامهنویسی سیستمی محسوب می‌شود. منظور از برنامهنویسی سیستمی، توسعه سیستم‌های عامل مانند ویندوز، کامپایلرها و درایورهای شبکه است. این زبان با این‌که از

¹² C Programming Language

¹³ Dennis Ritchie

محبوبیت زیادی برخوردار است ولی در عین حال برخی از افراد عقیده دارند که ساختار زبان سی برای یادگیری پیچیده است.

۱-۶-۴) تفاوت زبان برنامهنویسی پایتون با سی

جدول ۱-۱- مقایسه زبان‌های برنامهنویسی پایتون و سی

پایتون	سی
زبان برنامهنویسی پایتون اولین بار توسط خیدو فان روسوم در سال ۱۹۷۲ توسعه یافت.	زبان برنامهنویسی سی توسط دنیس ریچی در سال ۱۹۷۲ توسعه یافت.
اجرای برنامه‌های نوشته شده با پایتون کندرتر هستند.	اجرای برنامه‌های نوشته شده با سی سریع‌تر هستند.
مدیریت حافظه در پایتون بصورت خودکار انجام می‌شود. (نمی‌توان حافظه را مدیریت کرد).	مدیریت حافظه باید به صورت دستی در سی انجام شود. (نمی‌توان حافظه را مدیریت کرد).
پایتون یک زبان برنامهنویسی عمومی است (نمی‌توان برای کارهای مختلف از پایتون استفاده کرد).	زبان برنامهنویسی سی بیشتر برای توسعه برنامه‌های سختافزاری استفاده می‌شود.
کدهای پایتون با پسوند py. ذخیره می‌شوند.	کدهای سی با پسوند C. ذخیره می‌شوند.
یادگیری آسان	یادگیری نسبتاً دشوار

برنامه چاپ Hello World! در سی

```
C - Hello World  
  
#include <stdio.h>  
  
int main() {  
    printf("Hello World!");  
    return 0;  
}
```

برنامه چاپ Hello World! در پایتون

```
Python - Hello World  
  
print("Hello World!")
```

۱-۶-۵) آشنایی با زبان برنامه‌نویسی سی‌پلاس‌پلاس

زبان سی‌پلاس‌پلاس^{۱۴} یک زبان برنامه‌نویسی کامپیوترا می‌باشد که یک زبان شی‌گرا و سطح بالاست. اما به طور کلی به آن یک زبان سطح میانی می‌گویند چرا که هم قابلیت‌های یک زبان سطح بالا را دارد و هم قابلیت‌های یک زبان سطح پایین. این زبان برنامه‌نویسی بسیار قدرتمند است و از خانواده سی محسوب می‌شود.

با این زبان می‌توانید برنامه‌نویسی سیستم عامل، هسته و لایه‌های مختلف سیستم عامل را انجام دهید. با سی‌پلاس‌پلاس قادر هستید به تولید نرم‌افزارها، بازی‌سازی برای انواع کنسول‌ها، برنامه‌نویسی برای موبایل و تبلت و هم‌چنین برنامه‌نویسی ربات‌ها را انجام دهید. این زبان برنامه‌نویسی در صنایع پزشکی، فضایی، خودروهای هوشمند و اینترنت اشیا نیز کاربرد دارد.

۱-۶-۶) تفاوت زبان برنامه‌نویسی پایتون با سی‌پلاس‌پلاس

جدول ۱-۲- مقایسه زبان‌های برنامه‌نویسی پایتون و سی‌پلاس‌پلاس

C++	پایتون
سی‌پلاس‌پلاس معمولاً دارای خطوط طولانی کد است.	پایتون خطوط کد کمتری دارد.

^{۱۴} C Plus Plus (Cpp or C++) Programming Language

سی‌پلاس‌پلاس ساختار کد نسبتاً دشواری دارد.	پایتون ساختار کد آسانی دارد.
سی‌پلاس‌پلاس سرعت بالایی در اجرای کد دارد.	سرعت کمتری در اجرای کد دارد.
از کاربردهای سی‌پلاس‌پلاس می‌توان به توسعه وب، تجزیه و تحلیل سیستم عامل و ... اشاره کرد.	داده‌ها، محاسبات علمی و ... اشاره کرد.
کدهای سی‌پلاس‌پلاس با پسوند .cpp. ذخیره می‌شوند.	کدهای پایتون با پسوند .py. ذخیره می‌شوند.
یادگیری سی‌پلاس‌پلاس نسبتاً دشوار است.	یادگیری پایتون آسان است.

برنامه چاپ Hello World! در سی‌پلاس‌پلاس

```
● ● ● C++ - Hello World
#include <iostream>

int main() {
    std::cout << "Hello World!";
    return 0;
}
```

برنامه چاپ Hello World! در پایتون

```
● ● ● Python - Hello World
print("Hello World!")
```

۶-۶) آشنایی با زبان برنامه‌نویسی سی‌شارپ

بدون شک یکی از محبوب‌ترین و پرکاربردترین زبان‌های برنامه‌نویسی حال حاضر دنیا سی‌شارپ^{۱۵} می‌باشد و براساس آخرین تحقیقات صورت گرفته این زبان جزو پنج زبان برنامه‌نویسی برتر در دنیا می‌باشد. زبان سی‌شارپ، یک زبان برنامه‌نویسی مدرن است که از ویژگی شی‌ءگرایی برخوردار بوده و توسط شرکت مایکروسافت در سال ۲۰۰۰ طراحی شده است. ریشه این زبان، زبان سی است. به کمک سی‌شارپ، یک برنامه‌نویس به راحتی می‌تواند برنامه‌های مبتنی بر دسکتاپ ویندوز را ایجاد کند. نکته‌ی قابل توجه در مورد زبان سی‌شارپ که باعث برتری این زبان نسبت به سایر رقبا است، پشتوانه‌ای بزرگ شرکت مایکروسافت است. بر اساس گزارشی که در سال ۲۰۰۲

^{۱۵} Csharp(C#) Programming Language

منتشر شد، مشخص شد که شرکت مایکروسافت پس از صرف دو میلیون دلار هزینه و پنج میلیون ساعت کار بی‌وقفه توانسته این زبان برنامه‌نویسی را در اختیار توسعه دهنده‌گان در سراسر دنیا قرار دهد؛ به همین دلیل به جرات می‌توان گفت زبان برنامه‌نویسی سی‌شارپ آینده خوب و مطمئنی خواهد داشت.

۱-۶-۶) تفاوت زبان برنامه‌نویسی پایتون با سی‌شارپ

جدول ۱-۳- مقایسه زبان‌های برنامه‌نویسی پایتون و سی‌شارپ

سی‌شارپ	پایتون
ساختار کد نسبتاً دشواری دارد.	ساختار کد آسانی دارد.
دارای خطوط طولانی کد است	خطوط کد کمتری دارد
از کاربردهای آن می‌توان به توسعه وب، تجزیه و تحلیل دستکتاب و ... اشاره کرد.	از کاربردهای آن می‌توان به توسعه وب، تجزیه و تحلیل داده‌ها، هوش مصنوعی و ... اشاره کرد.
کدهای سی‌شارپ با پسوند CS. ذخیره می‌شوند.	کدهای پایتون با پسوند py. ذخیره می‌شوند
برنامه چاپ Hello World! در سی‌شارپ	

```
● ● ● C# - Hello World
using System;

namespace HelloWorld
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Hello World!");
        }
    }
}
```

برنامه چاپ Hello World! در پایتون

```
● ● ● Python - Hello World
print("Hello World!")
```

۱-۶-۷) آشنایی با زبان برنامه‌نویسی جاوا

زبان برنامه‌نویسی جاوا^{۱۶} در اوایل دهه ۱۹۹۰ میلادی توسط جیمز گاسلینگ^{۱۷} در شرکت سان‌مایکروسیستمز پایه‌ریزی شد. در حقیقت، نارضایتی گاسلینگ از اصول برنامه‌نویسی در زبان سی‌پلاس‌پلاس و نارسایی‌های این زبان موجب شد تا وی جاوا را بر مبنای زبان سی‌پلاس‌پلاس طراحی کند و بدین ترتیب توانست ایده‌های مد نظر خود را به نحو بهتر روی این زبان جدید عملی سازد. در حقیقت، جیمز گاسلینگ و سایر توسعه‌دهندگان این زبان برنامه‌نویسی از همان ابتدا شعار «یک بار بنویس، همه جا اجراش کن» را برای زبان جدید خود مد نظر قرار داده و در راستای دستیابی به هدفی متناسب با شعار این زبان نیز توانستند انقلابی در دنیای برنامه‌نویسی ایجاد کنند.

زبان برنامه‌نویسی جاوا یک زبان برنامه‌نویسی رایگان و شی‌گرا است که امروزه بسیاری از سازمان‌ها و شرکت‌های بزرگ در سراسر دنیا از آن استفاده می‌کنند. این زبان را امروزه می‌توان برای توسعه انواع اپلیکیشن‌های دسکتاب، اپلیکیشن‌های تحت وب و همین‌طور اپلیکیشن‌های مخصوص گوشی‌های هوشمند مورداستفاده قرارداد و از مزایای فوق العاده آن نهایت بهره را برد.

۱-۶-۷-۱) تفاوت زبان برنامه‌نویسی جاوا با پایتون

جدول ۱-۴- مقایسه زبان‌های برنامه‌نویسی پایتون و جاوا

جاوا	پایتون

^{۱۶} Java Programming Language

^{۱۷} James Gosling

پایتون کنترل از جاوا است اما برای برخی از وظایف به طور کلی جاوا سریع‌تر از پایتون است، به خصوص برای برنامه‌های کاربردی بزرگ و پیچیده. مانند اسکریپت‌نویسی و یادگیری ماشین به اندازه کافی سریع است.

جاوا کمی دشوارتر از پایتون برای یادگیری می‌باشد.

جاوا مناسب برای توسعه وب، علم داده، یادگیری ماشین، هوش مصنوعی، و اسکریپت‌نویسی. سیستم‌های تعبیه شده، و وب‌اپلیکیشن‌های سازمانی.

برنامه چاپ Hello World! در جاوا

```
Java - Hello World
class HelloWorld {
    public static void main(String args[])
    {
        System.out.println("Hello World");
    }
}
```

برنامه چاپ Hello World! در پایتون

```
Python - Hello World
print("Hello World!")
```

۱-۶-۸) آشنایی با زبان برنامه‌نویسی متلب

زبان برنامه‌نویسی متلب^{۱۸} یک زبان برنامه‌نویسی است و مجموعه کتابخانه‌های مخصوص خود را دارد. متلب مخفف عبارت Matrix Laboratory است. بهمین دلیل، در ابتدا به عنوان زبان برنامه‌نویسی ماتریس گرفته شده بود. متلب را کلیو مولر^{۱۹} خلق کرد. هدف او یافتن روشی جایگزین برای انجام‌دادن جبر خطی و محاسبات عددی، بدون نیاز به استفاده از فورترن، بود. بعدها در سال ۱۹۸۴ کلیو مولر، به همراه همکارانش، شرکت MathWorks را تأسیس کردند. این شرکت اولین نسخه‌ی رسمی خود از متلب را در سال ۱۹۸۴ منتشر کرد.

¹⁸ Matlab Programming Language

¹⁹ Cleve Moler

متلب یک زبان نسل چهارم (زبان‌هایی که به زبان محاوره نزدیک هستند)، با کارایی بالا و محیطی تعاملی برای محاسبات عددی، نمایش داده‌ها و برنامه‌نویسی است. با استفاده از متلب می‌توان الگوریتم‌های مختلفی را پیاده‌سازی کرد. می‌توانید داده‌ها را از منابع مختلف، مانند فایل‌ها، پایگاه‌داده‌ها یا وب، آپلود کنید و با متلب آن‌ها را تجزیه و تحلیل کنید و بعد با استفاده از طیف وسیعی از نمودارهای گرافیکی که در متلب وجود دارد آن‌ها را نمایش دهید؛ علاوه بر این، متلب کتابخانه‌ای از توابع ریاضی را شامل می‌شود که این را امکان می‌دهد که عملیاتی مانند جبر خطی و محاسبات ماتریس‌ها را انجام دهید.

۱-۸-۶) تفاوت زبان برنامه‌نویسی متلب با پایتون

جدول ۱-۵- مقایسه زبان‌های برنامه‌نویسی پایتون و متلب

متلب	پایتون
ساختار کد متلب مختصر و ریاضی محور است و برای عملیات ماتریسی و جبر خطی طراحی شده است.	ساختار کد پایتون خوانا و ساده است. این امر یادگیری و استفاده از آن را آسان‌تر می‌کند.
متلب برای محاسبات ریاضی و ماتریسی و وظایف مربوط به جبر خطی را سریعتر از پایتون است.	پایتون برای برخی از عملیات‌های ریاضی کندتر است، اما کتابخانه‌هایی مانند نامپای، می‌توانند سرعت انجام محاسبات را به طور قابل توجهی بهبود بخشند.
متلب عمدها برای محاسبات عددی، شبیه‌سازی مهندسی و تحقیقات علمی استفاده می‌شود.	پایتون یک زبان همه کاره است که در طیف گسترده‌ای از برنامه‌ها از جمله توسعه وب، اتوماسیون، هوش مصنوعی، علم داده و یادگیری ماشین استفاده می‌شود.
متلب یک نرم‌افزار تجاری است و برای استفاده از آن باید مجوز آن را خریداری کنید.	پایتون یک زبان منبع باز و رایگان است.

فصل دوم

آشنایی با زبان برنامه‌نویسی پایتون

۲) آشنایی با زبان برنامه‌نویسی پایتون و نصب و راهاندازی آن

پایتون یکی از زبان‌های برنامه‌نویسی پرطرفدار و قدرتمند است که به دلیل طراحی ساده و انعطاف‌پذیری بالا، در بسیاری از زمینه‌ها مورد استفاده قرار می‌گیرد. این زبان با هدف یادگیری آسان و کاربرد گستردۀ طراحی شده است و به همین دلیل به گزینه‌ای مناسب برای برنامه‌نویسان مبتدی و حرفه‌ای تبدیل شده است. ساختار شفاف و قابل فهم پایتون یکی از ویژگی‌های کلیدی آن است که باعث می‌شود افراد بتوانند بدون سردرگمی و با سرعت بیشتری کدنویسی را آغاز کنند.

پایتون با ارائه ساختارهای داده‌ای سطح بالا مانند فهرست‌ها، دیکشنری‌ها، مجموعه‌ها و غیره، برنامه‌نویسان را قادر می‌سازد تا داده‌ها را به‌طور کارآمد مدیریت و پردازش کنند. این ساختارها در کنار ابزارهای داخلی متنوع، توسعه برنامه‌های پیچیده را ساده‌تر می‌کنند. علاوه بر این، پایتون از یک رویکرد ساده اما مؤثر برای برنامه‌نویسی شی‌گرا استفاده می‌کند که به کاربران اجازه می‌دهد کدهای ماژولار و قابل توسعه‌تری ایجاد کنند.

یکی دیگر از ویژگی‌های برجسته پایتون تایپ پویا و اجرای تفسیری آن است. این خصوصیات امکان آزمایش و توسعه سریع برنامه‌ها را فراهم می‌کند و نیاز به کامپایل مجدد کد پس از هر تغییر را از بین می‌برند. به همین دلیل، پایتون انتخابی ایده‌آل برای پروژه‌هایی است که نیازمند بازخورد سریع یا توسعه سریع هستند، مانند اسکریپت‌نویسی، توسعه وب، و پردازش داده‌ها.

در نهایت، تنوع کاربردها و پشتیبانی از پلتفرم‌های مختلف، پایتون را به زبانی چندمنظوره تبدیل کرده است. از توسعه نرم‌افزارهای تجاری گرفته تا پروژه‌های علمی و تحقیقاتی، پایتون به دلیل سهولت استفاده و ابزارهای گستردۀ، در هر زمینه‌ای قابلیت‌های بی‌نظیری را ارائه می‌دهد. این ویژگی‌ها باعث شده‌اند که پایتون نه تنها در بین برنامه‌نویسان، بلکه در میان محققان، دانشمندان داده و توسعه‌دهندگان وب نیز محبوبیت بالایی کسب کند.

۲-۱) تاریخچه زبان برنامه‌نویسی پایتون

پایتون توسط خیدو فان روسوم در اواخر دهه هشتاد و اوایل دهه نود در موسسه تحقیقات ملی ریاضیات و علوم کامپیوتر در هلند توسعه یافت. در اوایل دهه ۱۹۹۰، خیدو به توسعه پایتون ادامه داد و در ۲۰ فوریه ۱۹۹۱، اولین نسخه عمومی پایتون را منتشر کرد. در طراحی زبان پایتون از چندین زبان برنامه‌نویسی دیگر از جمله ABC، Modula-3، سی و ... استفاده شد تا زبان پایتون یک ساختار منعطف و ساده داشته باشد.

در دورانی که خالق پایتون یعنی خیدو فان روسوم در حال دیدن شبکه BBC بود برنامه‌ای تلویزیونی به نام Monty Python که یک سریال کمدی دهه ۷۰ بود، در حال پخش بود. این برنامه ایده‌ای برای خیدو شد که نام چیزی را که خلق کرده است پایتون بگذارد و این‌گونه بود که داستان زبان برنامه‌نویسی پایتون آغاز شد. در سال ۱۹۸۲ خیدو به عنوان برنامه‌نویس تازه‌کار در موسسه تحقیقاتی مرکز ریاضیات و علوم کامپیوتری وارد تیم ABC شد. این تیم در مدت ۴ الی ۵ سال بر روی پروژه‌ای کار می‌کرد تا بتواند زبان برنامه‌نویسی جدیدی خلق کند که قدرت، خوانایی، ظرافت و سادگی را داشته باشد. اما این تیم به موفقیت چشم‌گیری نرسید و پروژه مختومه شد. پس از شکست تیم ABC، خیدو به تیم Amoeba در موسسه تحقیقاتی CWI پیوست و کار خود را بر روی توسعه سیستم‌عامل آغاز کرد. اما پس از مدتی مدیر پروژه در دانشگاه به مقام استادی رسید و اعضای تیم از هم جدا شدند. پس از آن خیدو به تیم مالتی مедیا در CWI وارد شد. قرارگیری خیدو در دو تیم ABC و Amoeba باعث شد که از پروژه‌های آن‌ها الگو برداری کند. از همان سالی که خیدو با گروه ABC کار می‌کرد در فکر تحقق بخشیدن به پروژه تیم ABC بود؛ این که بتواند زبانی ساده، با طراحی بهتر از زبان قدرتمندی نظری سی پیاده‌سازی کند. ترکیب ABC و انگیزه توسعه و رشد سیستم‌عامل Amoeba، سبب شد تا خیدو فان روسوم به دنبال تحقیق رویای خود برود و تبدیل به خالق پایتون شود.

زمان کریسمس سال ۱۹۸۹ در هلند بود. خیدو در خانه نشسته بود و درحال گذراندن تعطیلات کریسمس خود بود. او با پروژه محبوب خود سرگرم شد تا اوقات فراغت خود را در تعطیلات کریسمس با فعالیت مورد علاقه‌اش

بگذراند. خیدو چون از ABC ایده گرفته بود، خواست نام آن را B بگذارد اما در آن زمان، زبانی با این نام وجود داشت. همان‌طور که گفته شد خیدو با دیدن برنامه محبوبش تصمیم گرفت آن را پایتون نام‌گذاری کند. نامی از برنامه Monty Python's Flying Circus گرفته شد. در اوخر دسامبر ۱۹۸۹ کار خود را به صورت جدی‌تر دنبال کرد و حدود یک سال بعد، در سال ۱۹۹۰ توانست اولین نسخه پایتون را خلق کند.

۲-۲) ویژگی‌های زبان برنامه‌نویسی پایتون

پایتون دارای ویژگی‌های فراوان و قدرتمندی است که این ویژگی‌ها توانستند پایتون را به یکی از محبوب‌ترین و ارزشمندترین زبان‌های برنامه‌نویسی در جهان تبدیل کند. در جدول ۲-۱، به بررسی مهم‌ترین ویژگی‌های زبان برنامه‌نویسی پایتون پرداخته شده است.

جدول ۲-۱- بررسی ویژگی‌های زبان برنامه‌نویسی پایتون

ویژگی	توضیحات
سادگی و خوانایی	یکی از ویژگی‌های برجسته پایتون، سادگی در نوشتار و خوانایی کد است. از این زبان می‌توان به عنوان "زبانی که با انسان‌ها صحبت می‌کند" ^{۲۰} یاد کرد. ساختار کد، ساده و مشخص دارد که امکان نوشتن کدهای تمیز و قابل فهم را فراهم می‌کند و از آن به عنوان یک زبان برنامه‌نویسی کاربر پسند ^{۲۰} یاد می‌کنند.
جامعه فعال و گستردگی	زبان پایتون یک جامعه بزرگ از برنامه‌نویسان و متخصصان دارد که به توسعه و پشتیبانی از این زبان مشغول‌اند. این ویژگی باعث می‌شود که مشکلات پیش آمده به آسانی برطرف شود.

²⁰ User Friendly

منبع باز و رایگان

پایتون یک زبان منبع باز است؛ به این معنا که همه می‌توانند از کد منبع آن به صورت آزاد استفاده

کنند و در ایجاد قابلیت‌های جدید و توسعه آن مشارکت داشته باشند.

سازگاری با زبان‌های

پایتون سازگاری خوبی با زبان‌های دیگر دارد. شما می‌توانید از کد زبان‌های دیگر مانند سی شارپ در

کد پایتون خود استفاده کنید. هم‌چنین می‌توانید کد پایتون خود را مابین کد زبان‌های دیگر مانند سی

دیگر

شارپ قرار دهید.

ماژول‌ها و

پایتون، ماژول‌ها و کتابخانه‌های فراوانی دارد؛ بنابراین شما مجبور نیستید برای هر بخشی از پروژه‌تان،

کدنویسی کنید چراکه می‌توانید کدهایی را که از قبل در کتابخانه‌ها وجود دارد، استفاده کنید و این

کتابخانه‌های فراوان

باعث افزایش سرعت کدنویسی و صرفه‌جویی در زمان می‌شود.

از ویژگی دیگر این زبان، می‌توان به سطح بالا بودن آن اشاره کرد. منظور از سطح بالا بودن یعنی این که

به زبان انسان نزدیک است و هر کسی بدون داشتن دانش برنامه‌نویسی، می‌تواند کدها را بخواند و تا

حدودی متوجه شود.

پورتابل و قابل حمل

منظور از قابل حمل بودن این است که زبان پایتون بر روی اکثر سیستم‌عامل‌ها (ویندوز، لینوکس و

مک) اجرا می‌شود و می‌توان کدها را بدون اعمال تغییرات، ببروی انواع سیستم‌عامل‌ها اجرا کرد.

بودن

زبان برنامه‌نویسی

پایتون یک زبان تفسیر شده است. منظور از مفسری این است که کدها به صورت خط به خط اجرا

می‌شود و در نتیجه اگر خطایی وجود داشته باشد، می‌توانیم به آسانی خط را مشاهده و برطرف کنیم.

مفسری

پایتون یک زبان برنامه‌نویسی شی‌گرا است و از ویژگی‌هایی مانند وراثت، چندشکلی و غیره پشتیبانی

می‌کند. این ویژگی به برنامه‌ها اجازه می‌دهد در درازمدت کارآمد باشند و با تعریف اشیای مختلف،

شی‌گرایی

سیستم نرم افزاری را مدلسازی کرد.(در بخش دوم کتاب، شی‌گرایی توضیح داده می‌شود.)

کاربردهای متنوع

می‌توانید از پایتون در هر پروژه کوچک و بزرگ استفاده کنید (از نوشتن یک خط کد ساده گرفته تا

نوشتن الگوریتم‌های هوش مصنوعی).

فرصت های شغلی

پایتون یک زبان بسیار محبوب در بازار کار است. یادگیری پایتون می تواند چندین فرصت شغلی در علم

داده، هوش مصنوعی، توسعه وب و موارد دیگر ایجاد کند.

۳-۲) کاربردهای زبان برنامهنویسی پایتون

تا به اینجا درباره زبان پایتون و ویژگی های این زبان قدرتمند آشنا شدید. حال اولین چیزی که شما باید بدانید این است که کاربرد پایتون در کجاهاست و چرا باید پایتون را یاد گرفت.

زبان برنامهنویسی پایتون کاربردهای بسیاری دارد و از آن به عنوان زبان همه کاره یاد می شود. از پایتون می توان برای ساخت پروژه های کوچک و بزرگ استفاده کرد که همین ویژگی ها و کاربردها، دلیلی برای محبوبیت پایتون شده است. در جدول ۲-۲، به بررسی کاربردهای زبان برنامهنویسی پایتون پرداخته شده است.

جدول ۲-۲- بررسی کاربردهای زبان برنامهنویسی پایتون

کاربردها	توضیحات
هوش مصنوعی	امروزه هوش مصنوعی و یادگیری ماشین جزء مباحث روز دنیا می باشد. پایتون با کتابخانه ها و ابزارهای داخلی خود توسعه الگوریتم های هوش مصنوعی را آسان کرده است. علاوه بر این، کدهای ساده، مختصرا و قابل خواندن را ارائه می دهد که نوشتن الگوریتم های پیچیده برای توسعه دهندگان را آسان تر می کند.
یادگیری ماشین	از ابزارهای پایتون که برای هوش مصنوعی استفاده می شود، می توان به پایتورچ، کراس، تنسورفلو و ... اشاره کرد.
توسعه برنامه های تحت وب	توسعه وب یکی از مهم ترین کاربردهای پایتون است؛ پایتون طیف گسترده ای از فریمور کها و کتابخانه هایی نظیر جنگو، فلسک و موارد دیگر ارائه می دهد که امکان سهولت را برای توسعه دهندگان فراهم می کند. معمولا از پایتون برای برنامه نویسی سمت سرور استفاده می شود.

برنامه‌های محبوب مانند گوگل، نتفلیکس و ردیت همگی از پایتون استفاده می‌کنند. پایتون اغلب

توسعه نرم‌افزار به عنوان یک زبان پشتیبانی برای توسعه دهنده‌گان نرم افزار، برای کنترل و مدیریت ساخت و تست استفاده می‌شود.

تجزیه و تحلیل پایتون توانایی تجزیه و تحلیل و تجسم داده‌ها در قالب نمودارها را دارد و از ابزارهایی نظیر پانداس، نامپای، Matplotlib داده‌ها می‌توان این کارها را انجام داد.

توسعه بازی پایتون در توسعه بازی، در درجه اول برای ساخت بازی‌های دو بعدی استفاده می‌شود. کتابخانه‌هایی مانند پای‌گیم ابزارهای مورد نیاز برای ساخت بازی‌ها را در اختیار توسعه‌دهنده‌گان قرار می‌دهند.

برنامه‌های دسکتاپ پایتون به توسعه‌دهنده‌گان کمک می‌کند تا رابط کاربری گرافیکی^{۲۱} را به راحتی توسط کتابخانه‌ها و فریمورک‌هایی نظیر پای‌کیودی، کیوی و ... ساخته شود.

استخراج داده از وب اسکرپینگ یا استخراج داده یک فرآیند خودکار است که برای استخراج اطلاعات از وبگاه‌ها به روشنی ساده و سریع انجام می‌گیرد. از ابزارهای پایتون نظیر اسکرپی و سلنیوم برای استخراج اطلاعات از اینترنت وبگاه‌ها استفاده می‌شود.

پردازش تصویر این امروزه پردازش تصویر یکی از زمینه‌هایی است که به سرعت در حال رشد است. پردازش تصویر در صنایع مختلفی از جمله بینایی کامپیوتر، تصویربرداری پزشکی، امنیت و ... استفاده می‌شود. پایتون نیز برای پردازش تصویر، کتابخانه و فریمورک‌هایی نظیر اوپن‌سی‌وی و ... دارد که کار پردازش را انجام می‌دهد.

امنیت و تست نفوذ پایتون به دلیل ساختار ساده و قدرتمند به یکی از زبان‌های محبوب در حوزه هک و امنیت تبدیل شده است. از پایتون می‌توان در زمینه تست نفوذ و ارزیابی امنیت، توسعه بدافزار، مهندسی اجتماعی، باگ‌بانی و ... استفاده کرد.

²¹ Graphical User Interface (GUI)

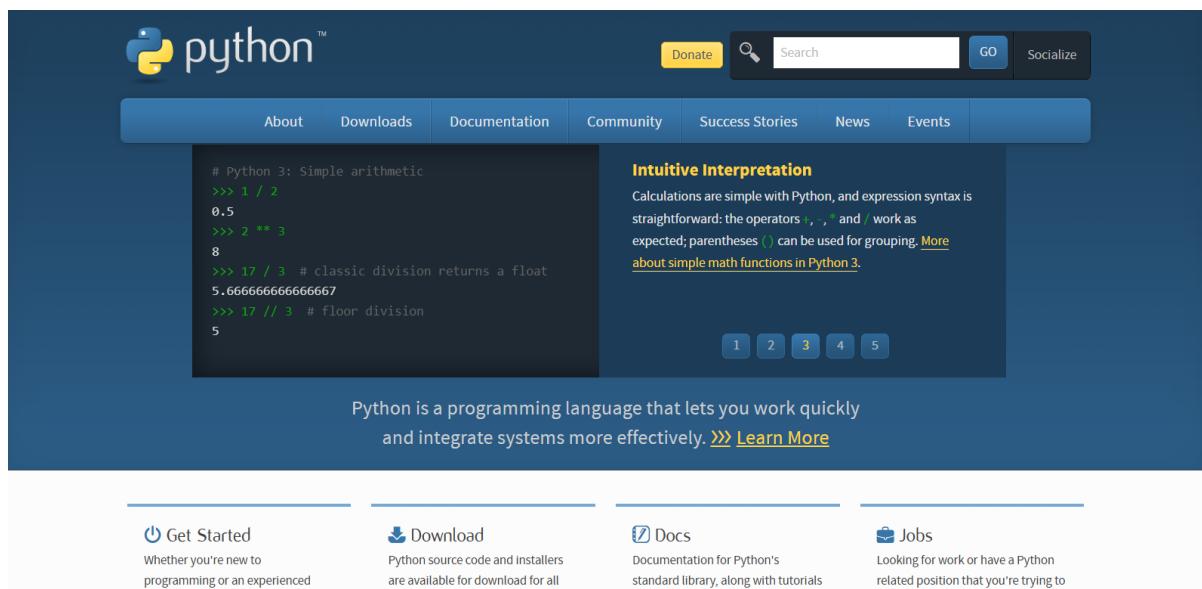
۴-۲) جایگاه زبان پایتون در شرکت‌های بزرگ دنیا

- هنگامی که گوگل تازه شروع به کار کرده بود، پایتون را به دلیل ماهیت ساده استفاده کرد و از آن زمان تاکنون به استفاده از آن ادامه داده است. گوگل از این زبان در موتور جستجو، یوتیوب، یادگیری ماشین، هوش مصنوعی، پروژه‌های رباتیک و موارد دیگر استفاده می‌کند.
- امروزه، زبان پایتون هسته فیسبوک است و ۲۱٪ از کل پایگاه کد را تشکیل می‌دهد. این یک جزء کلیدی در سیستم فیسبوک است که اطمینان می‌دهد که پلتفرم آماده و در حال اجرا است.
- در حال حاضر اینستاگرام، یکی از بزرگ‌ترین شبکه‌های مجازی در جهان است که برنامه‌نویسی سمت سرور آن از جنگو که یکی فریمورک‌های پایتون می‌باشد، نوشته شده است.
- اسپاتیفای از پایتون در جنبه‌های مختلف مدیریت زیرساخت خود استفاده می‌کند. از نظارت بر سلامت سیستم گرفته تا سیستم‌های هوش مصنوعی پیشنهاد دهنده.
- نتفلیکس شبکه تلویزیونی اینترنتی با بیش از ۱۴۰ میلیون عضو است. پایتون تقریباً در بیشتر بخش‌های نتفلیکس استفاده می‌شود، از سیستم پیشنهاد دهنده گرفته تا مدیریت CDN برای ارائه محتوا ویدیویی.
- آمازون عادات خرید و الگوی خرید مشتریان خود را تجزیه و تحلیل می‌کند تا پیشنهادهای دقیق به آن‌ها ارائه دهد. این امر توسط ابزارهای هوش مصنوعی پایتون و پایگاه داده آمازون انجام می‌شود.
- ناسا نیز از پایتون برای محاسبات علمی، تجزیه و تحلیل داده‌ها و تجسم استفاده می‌کند. به عنوان مثال، پروژه سیستم هشدار گرما، از پایتون برای تجزیه و تحلیل پارامترهای دما و تجسم نقشه‌های حرارتی استفاده می‌کند.
- اگرچه بسیاری از کتابخانه‌های داخلی دراپ باکس به صورت اختصاصی است و منبع‌باز نیست، اما این شرکت یک API بسیار قدرتمند با کد پایتون را راهاندازی کرده است. هم‌چنین توسعه‌دهندگان دراپ باکس از پایتون در بیشتر برنامه‌نویسی‌های سمت سرور خود استفاده می‌کنند.

۲-۵) نصب و راه اندازی پایتون

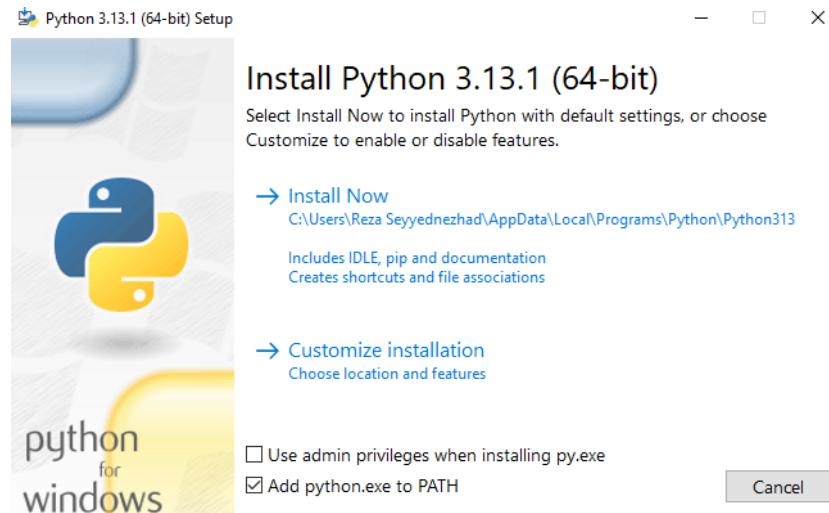
۲-۵-۱) نصب روی سیستم عامل ویندوز

احتمال اینکه زبان برنامه‌نویسی پایتون، به طور پیش‌فرض، روی کامپیوترهای با سیستم عامل ویندوز نصب شده باشد، بسیار پایین است و معمولاً از قابلیت نصب پایتون به صورت پیش‌فرض برخوردار نیستند. خوشبختانه، نصب پایتون روی سیستم عامل ویندوز بسیار راحت است و تنها کافی است آخرین نسخه پایتون را از لینک [/https://www.python.org](https://www.python.org) دانلود کنید.



شکل ۲-۱- صفحه‌ی اصلی وبگاه رسمی پایتون

در سیستم عامل ویندوز، این امکان وجود دارد که یکی از پردازنده‌های ۶۴ بیتی یا ۳۲ بیتی را روی سیستم خود نصب کنید. در صورتی که مطمئن نیستید چه نوع پردازنده‌ای رو سیستم شما نصب شده است، نصب پایتون ۶۴ بیتی مطمئن‌تر خواهد بود. بعد از دانلود فایل نصبی پایتون، آن را طبق مراحل زیر نصب کنید.

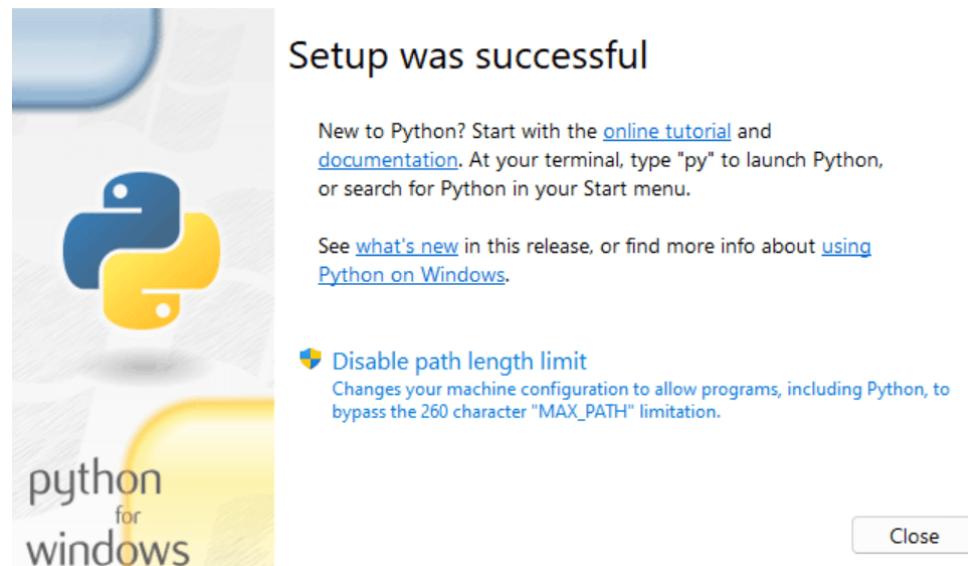


شکل ۲-۲- صفحه‌ی اصلی فایل پایتون دانلود شده از وبگاه پایتون

برای نصب پایتون، طبق شکل ۲-۲، گزینه Install Now را انتخاب کنید.

نکته: ضروری است تیک گزینه Add python.exe to PATH مطابق با شکل ۲-۲ را فعال کنید.

نکته: هنگام فرایند نصب، بسته‌ها و سایر فایل‌های پیش‌فرض پایتون در سیستم نصب خواهند شد. هنگامی که نصب با موفقیت انجام شد، شکل ۳-۲ ظاهر می‌شود.



شکل ۲-۳- صفحه‌ی مربوط به موفقیت آمیز بودن نصب پایتون

برای بررسی اینکه آیا پایتون در سیستم نصب شده است یا خیر، باید موارد زیر را انجام دهید.

۱. ترمینال یا CMD را در ویندوز خودتان باز کنید.
۲. سپس با زدن دستور زیر اطمینان حاصل کنید که پایتون نصب شده است.

python --version

نکته: اگر پایتون نصب شده باشد، ورژن نصبی پایتون نمایش داده خواهد شد.

نکته: اگر پایتون نصب نشده باشد پیغام زیر نمایش داده می‌شود و باید دوباره نصب شود:

“python' is not recognized as an internal or external command, operable program or batch file.”

۲-۵-۲) نصب روی سیستم عامل مک

دو روش اصلی برای نصب پایتون در سیستم عامل مک وجود دارد:

۱. نصب از وبگاه پایتون: این سریع‌ترین روش برای نصب پایتون است. سپس مراحل زیر را دنبال کنید:

- به وبگاه پایتون به آدرس <https://www.python.org/downloads/> بروید.
- در بخش Downloads، نسخه مناسب پایتون را برای سیستم عامل مک خود را انتخاب و دانلود کنید. توجه داشته باشید که دو نسخه برای انتخاب وجود دارد، یکی برای پردازنده های Intel و دیگری برای پردازنده های Apple M1 و باید نسخه نصبی موردنظر را دانلود کرده و نصب کنید.

- فایل نصبی دانلود شده را اجرا کنید و طبق دستورالعمل هایی که برای نصب روی ویندوز توضیح داده شد، عمل کنید.
 - برای اطمینان از نصب شدن پایتون دستور `python3 --version` در ترمینال وارد کنید. اگر پایتون نصب شده باشد؛ `python` به همراه نسخه نصب شده نمایش داده می شود.
- نصب با Homebrew .^۲
- یک مدیر بسته محبوب برای سیستم عامل مک است که می تواند برای نصب آسان بسیاری از برنامه ها، از جمله پایتون، استفاده شود. اگر Homebrew را نصب ندارید، می توانید آن را با دستور زیر نصب کنید:

```
/bin/bash -c "$(curl -fsSL https://brew.sh/)"
```

- پس از نصب Homebrew، می توانید پایتون را با دستور زیر نصب کنید. این دستور آخرین نسخه پایتون را نصب می کند:

```
brew install python
```

- همچنین می توانید نسخه خاصی از پایتون را با استفاده از دستور زیر نصب کنید (برای مثال نسخه ۳.۱۰):

```
brew install python@3.10
```

▪ نکته: اگر قبل از نسخه‌های قبلی پایتون را نصب کرده‌اید، می‌توانید با استفاده از دستور زیر آن را

به روز کنید:

brew upgrade python

▪ پس از نصب پایتون، می‌توانید با باز کردن برنامه Terminal و وارد کردن دستور زیر، نسخه

پایتون نصب شده را بررسی کنید:

۳-۵-۲) معرفی نرم‌افزارهایی جهت کدنویسی پایتون

برای کدنویسی پایتون، می‌توانید از نرم‌افزارهای مختلف استفاده کنید.

۱. ویرایشگرهای کد: نرم‌افزارهای ساده و کاربردی هستند که برای نوشتن و ویرایش کدها استفاده می‌شوند.

برخی از ویرایشگرهای کد برای پایتون عبارتند از:

❖ VS Code ویرایشگر کد منبع باز و محبوب است که از بسیاری از زبان‌های برنامه‌نویسی از جمله

پایتون پشتیبانی می‌کند. این ویرایشگر دارای افزونه‌ها و ویژگی‌های بسیاری است که می‌تواند

به شما در کدنویسی پایتون کمک کند.

❖ سابلایم تکست^{۲۲} ویرایشگر کد دیگری است که از بسیاری از زبان‌های برنامه‌نویسی از جمله پایتون

پشتیبانی می‌کند. این ویرایشگر دارای رابط کاربری ساده است و می‌تواند با افزونه‌های مختلف

کدنویسی را آسان‌تر سازد.

²² Sublime Text

۲. محیط‌های توسعه یکپارچه^{۲۳}: نرمافزارهای پیشرفته‌تری نسبت به ویرایشگرهای کد هستند و ویژگی‌های

بیشتری برای کمک به شما در توسعه نرمافزارها را ارائه می‌دهند. برخی از این محیط‌ها برای پایتون

عبارتند از:

❖ پایچارم^{۲۴} توسط شرکت JetBrains توسعه یافته است. پایچارم دارای ویژگی‌های بسیاری است

که می‌تواند در کدنویسی به شما کمک کند، مانند تکمیل کد، تجزیه و تحلیل کد، اشکال‌زدایی

و بازنویسی.

❖ اسپایدر^{۲۵} منبع‌باز است و دارای ویژگی‌های بسیاری است که می‌تواند به شما در کدنویسی علمی

و محاسباتی با پایتون کمک کند، از جمله ابزارهای تجزیه و تحلیل داده.

نکته: انتخاب ویرایشگر کد مناسب برای شما به نیازهای شما بستگی دارد. اگر تازه‌کار هستید، VS Code گزینه‌ی

خوبی برای شروع می‌باشد.

چند نکته اضافی برای استفاده از ویرایشگر کد برای اجرای کدهای پایتون:

- هنگامی که فایل پایتون خود را ذخیره می‌کنید، باید از پسوند .py استفاده کنید. به عنوان مثال، اگر فایلی

که کدهای شما در آن قرار دارد، app نام‌گذاری کردید، باید آن را به صورت app.py ذخیره کنید.

- بعد از این‌که کد خود را نوشته‌ید و فایلتان را ذخیره کردید، ترمینال یا CMD را باز کرده و به محلی که

کد خودتان را ذخیره کردید، رفته و برای اجرای کد خود از دستور زیر در ترمینال استفاده کنید. به عنوان

مثال، برای اجرای فایل app.py، از دستور زیر در CMD استفاده کنید.

python3 app.py

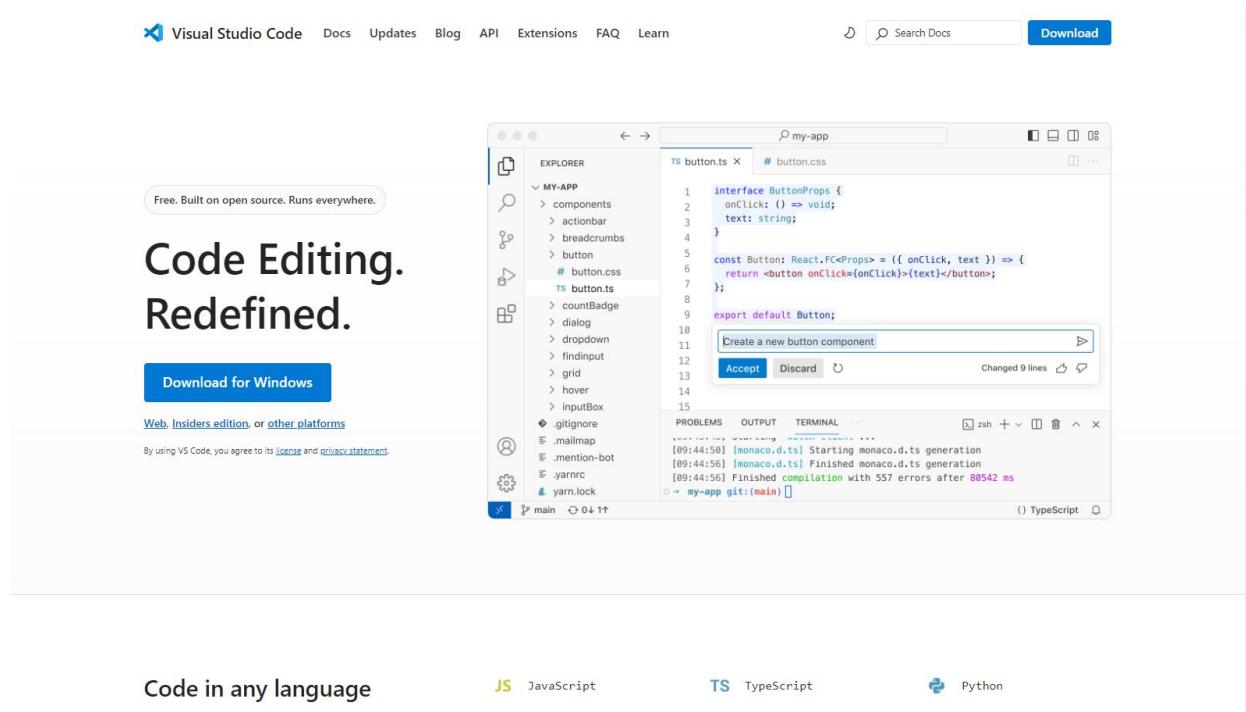
²³ Integrated Development Environment (IDE)

²⁴ Pycharm

²⁵ Spyder

اگر از VS Code استفاده کنید نیاز نیست که از ترمینال یا CMD برای اجرای فایل پایتون استفاده کنید. با استفاده از ترمینال خود VS Code، می‌توانید کدهای پایتون را اجرا کنید. در این کتاب از ویرایشگر VS Code جهت اجرای کدهای پایتون استفاده می‌شود.

برای دانلود VS Code، ابتدا از وبگاه <https://code.visualstudio.com/download> فایل موردنیاز را طبق سیستم‌عاملی که دارید دانلود کرده و آن را نصب کنید.

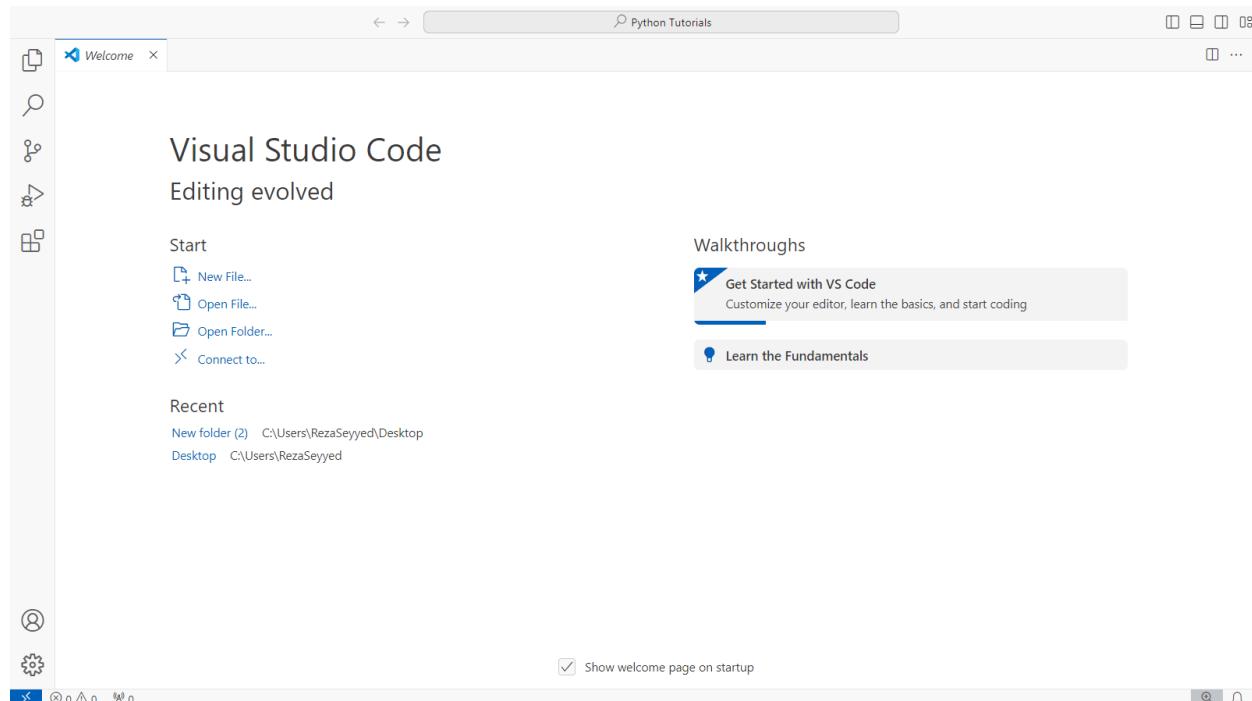


شکل ۲-۴- صفحه‌ی اصلی وبگاه رسمی VS Code

نکته: فرایند نصب آسان و همانند نصب پایتون می‌باشد. هنگام نصب VS Code تیک گزینه Add to PATH را فعال کنید.

۶-۲) آشنایی با محیط VS Code و نصب افزونه‌های پایتون

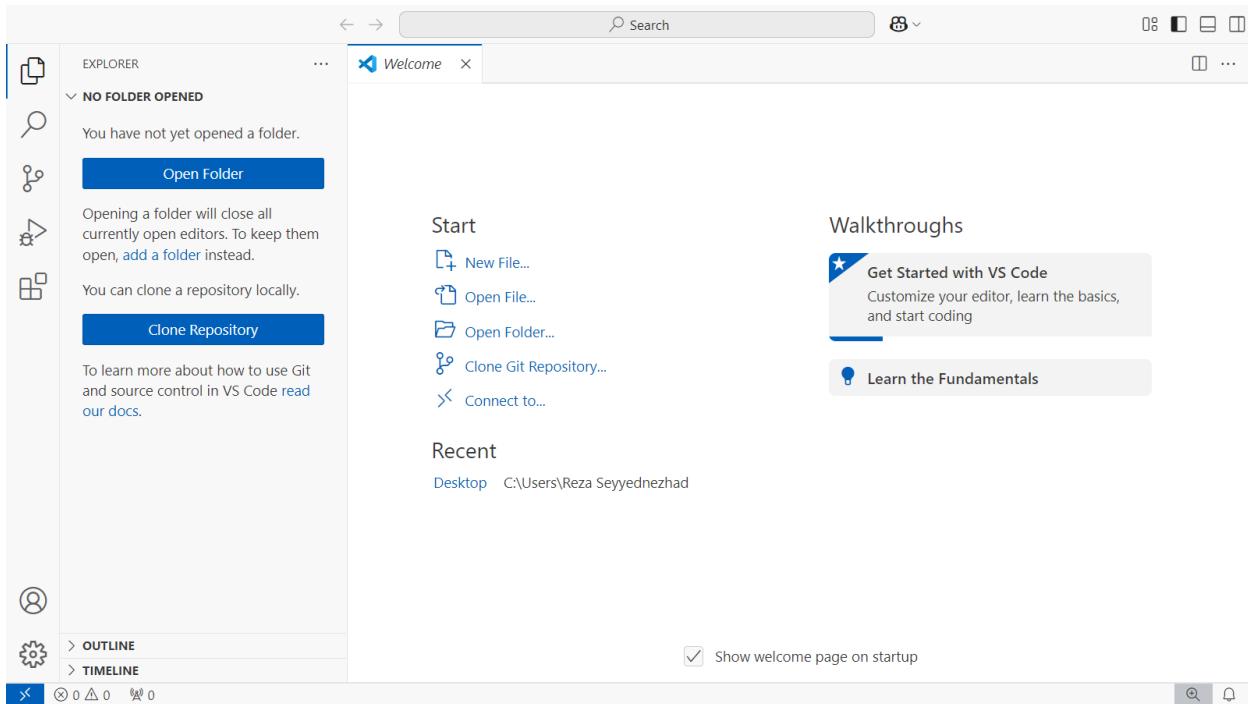
هنگامی که نرم‌افزار VS Code را باز می‌کنید با محیطی نظیر شکل ۶-۵ روبرو می‌شوید که می‌توانید کدهای خود را در آنجا نوشته، ذخیره و اجرا کنید.



شکل ۶-۵-محیط کلی از نرم‌افزار *VS Code*

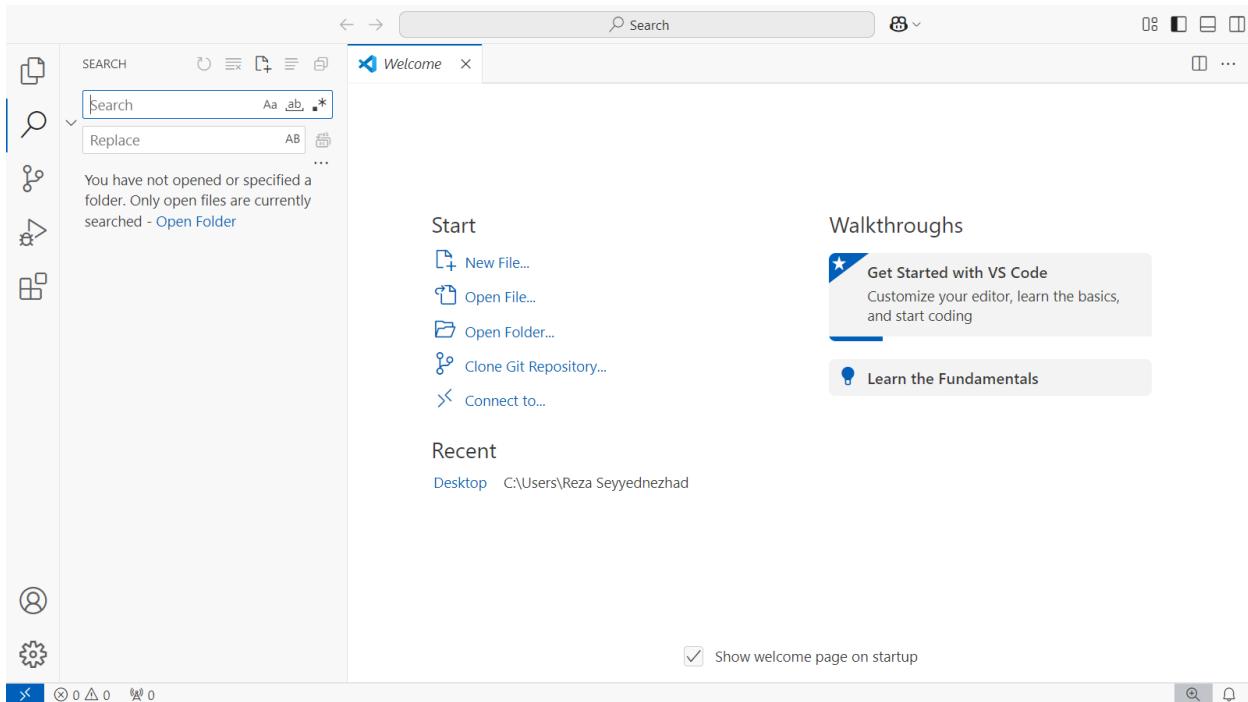
در نوار سمت چپ چندین آیکون وجود دارد که هر کدام مربوط به بخش خاصی از محیط نرم‌افزار مربوط می‌شود.

همانطور که در شکل ۶-۲ مشاهده می‌کنید، آیکون اول مربوط به قسمت Explorer می‌باشد که در این قسمت، فolderها و فایل‌ها قرار می‌گیرند.



شکل ۲-۶- آیکون مربوط به قسمت *Explorer* در نرم افزار *VS Code*

در شکل ۲-۷، آیکون دوم مربوط به جستجوی فایل مورد نظر می باشد. اگر تعداد فایل هایی که در Explorer ایجاد کردید زیاد باشد، از قسمت Search می توانید، فایل مورد نظرتان را جستجو کنید.

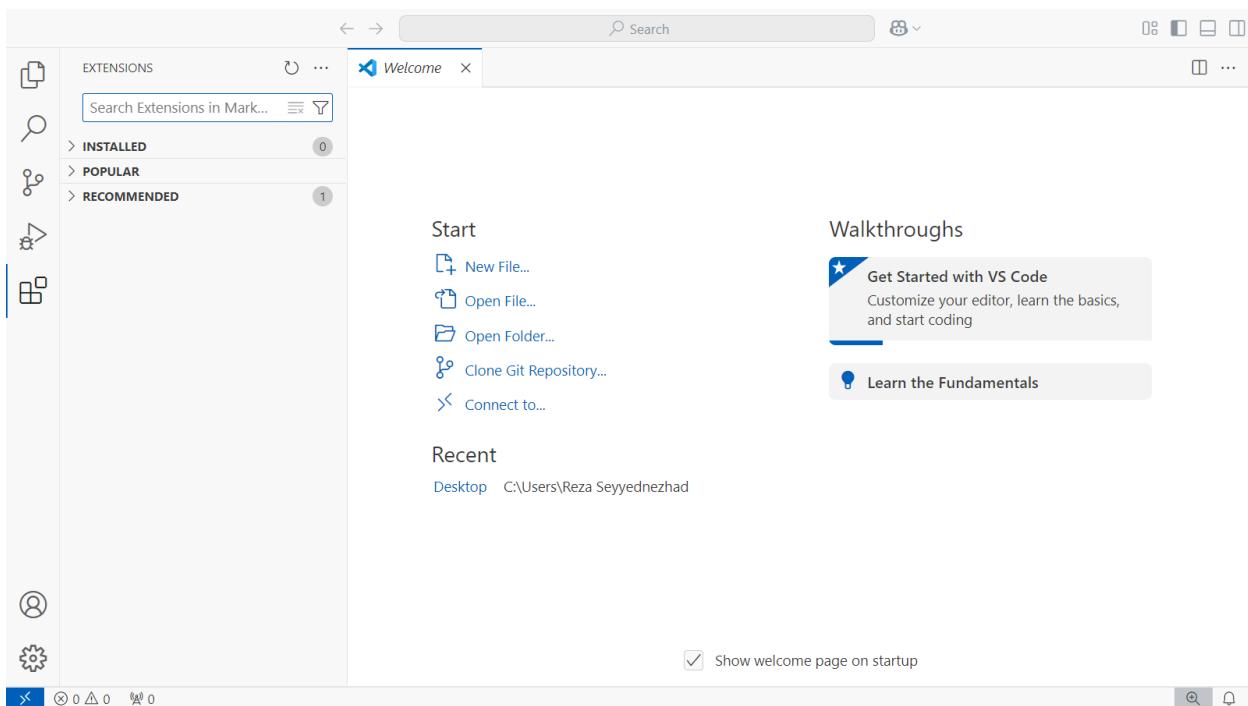


شکل ۲-۷-آیکون مربوط به قسمت *Search* در نرم افزار *VS Code*

آیکون سوم مربوط به فرایندهای Git و سیستم کنترل نسخه می‌باشد که جزء سرفصل‌های این کتاب نمی‌باشد.

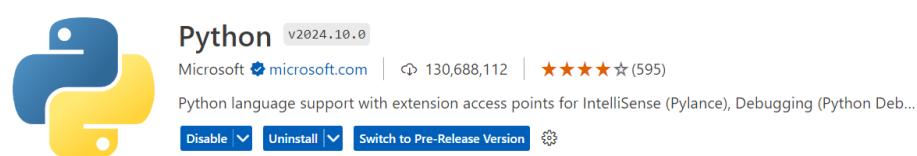
آیکون چهارم مربوط به اجرا و دیباگینگ کدها می‌باشد که در آینده درمورد همین قسمت توضیحات بیشتر ارائه خواهد شد.

آیکون پنجم طبق شکل ۲-۸، مربوط به افزونه‌ها می‌باشد. معمولاً برای کدنزی آسان و سریع و همچنین برخی تغییرات محیط VS Code از این قسمت استفاده می‌کنند و افزونه‌های مورد نظر را نصب می‌کنند.



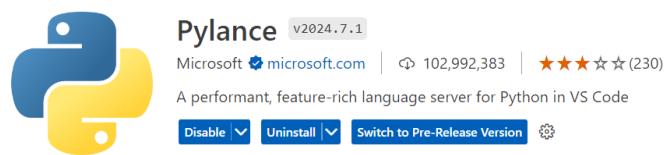
شکل ۲-۸-آیکون مربوط به قسمت *Extensions* در نرم افزار *VS Code*

نکته: قبل از اجرای برنامه های پایتون، از قسمت افزونه ها در نرم افزار *VS Code*، چند افزونه را در نصب کنید. این افزونه ها برای کدنی سریع و آسان و همچنین اجرای کدهای پایتون نیاز و ضروری است.



شکل ۲-۹-نصب افزونه *Python* از قسمت *Extension* در *VS Code*

Pylance •



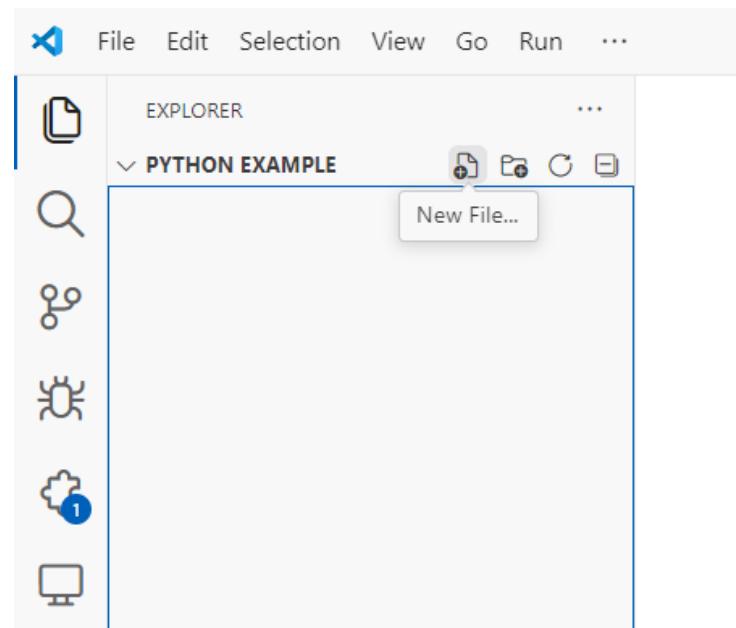
شکل ۲-۱۰- نصب افزونه *Pylance* در *VS Code Extension* از قسمت *Pylance*



شکل ۲-۱۱- نصب افزونه *Python Indent* در *VS Code Extension* از قسمت *Python Indent*

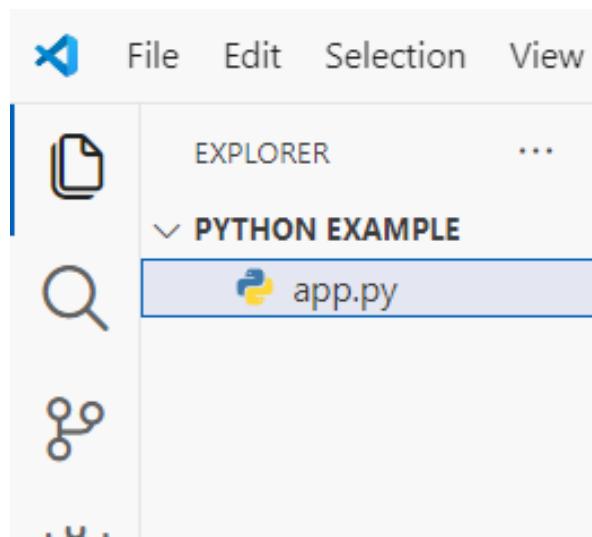
۷-۲) اجرای کدهای پایتون با *VS Code*

برای این که بتوانید کدهای پایتون را در یک فایلی بنویسید و آن فایل را در یک فolderی ذخیره کنید، ابتدا باید در یک درایوی یک پوشه ایجاد کرده و پوشه نامگذاری کنید (نام پوشه باید انگلیسی باشد). سپس روی آن پوشه راست کلیک کرده و در نواربار ظاهر شده گزینه Open With Code (آیکون vs code) انتخاب کرده و منتظر باز شده VS Code بمانید. سپس طبق شکل ۱۲-۲ از نوار سمت چپ VS Code، روی دکمه «New File» کلیک کنید تا یک فایل جدید ایجاد شود.



شکل ۲-۱۲- نحوه ایجاد فایل در *VS Code*

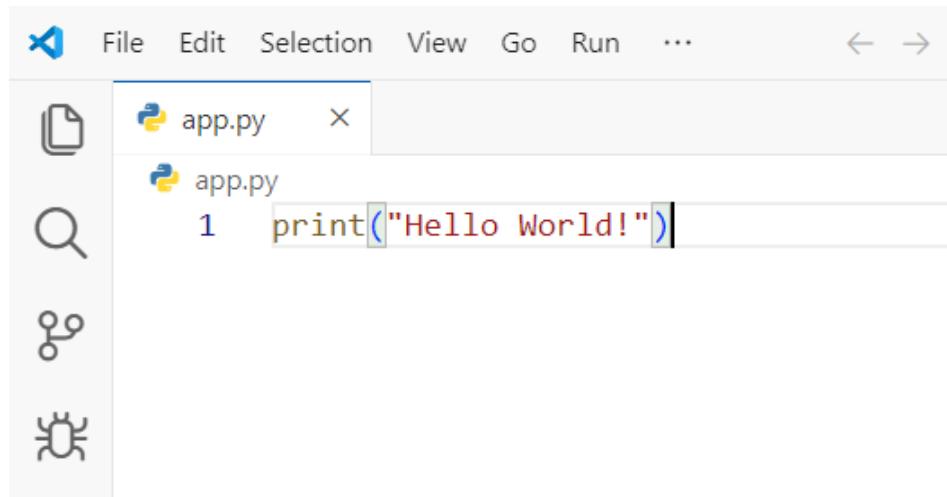
برای فایل ایجاد شده یک نام دلخواه نوشته و با پسوند `.py`. آن را ذخیره کنید (برای مثال `myApp.py`).



شکل ۲-۱۳- فایل پایتونی ایجاد شده در *VS Code*

بعد از اینکه فایل پایتون همانند شکل ۲-۱۳ ایجاد شد، زمان نوشتن اولین برنامه فرا رسیده است.

برنامه Hello World یک برنامه ساده است که عبارت "Hello, World!" را به چاپ می‌رساند. برای اجسام این کار باید از دستور print طبق شکل ۱۴-۲ استفاده کنید.



شکل ۱۴-۲- نوشتن برنامه چاپ "Hello World!" در *VS Code*

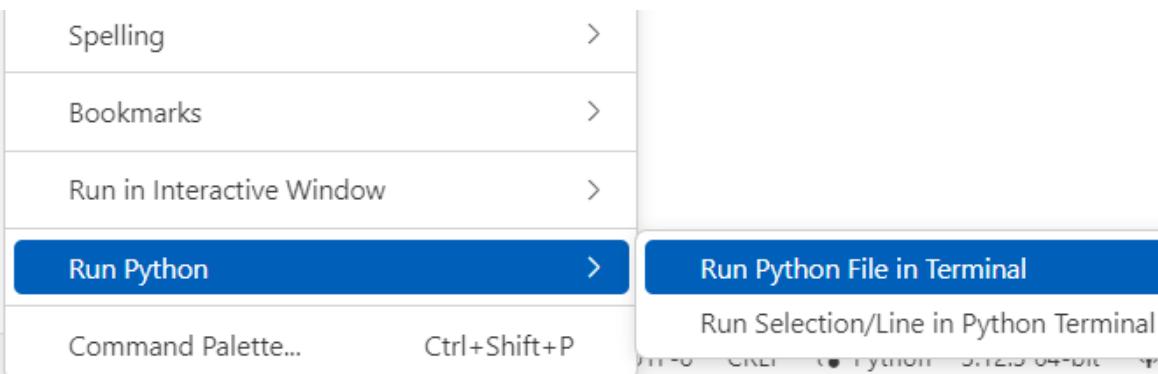
توضیحات شکل ۱۴-۲:

- دستور print برای چاپ متن در کنسول استفاده می‌شود.
- عبارت Hello World! متنی است که چاپ می‌شود.

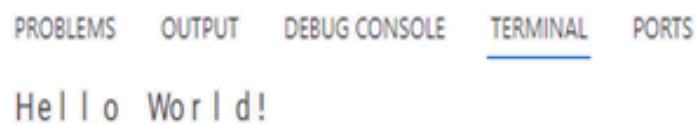
بعد از نوشتن دستور، فایل را بصورت دستی ذخیره کرده و یا با کلید Ctrl + S فایل خود را ذخیره کنید.

سپس در محیط VS Code فایل خود را به صورت زیر اجرا کنید:

روشی ساده برای اجرای کدهای پایتون با VS Code، استفاده از گزینه Run Python می‌باشد. با راست کلیک کردن روی کدها، نوار ابزاری ظاهر می‌شود. از قسمت Run Python File in روی گزینه Terminal کلیک کنید.



شکل ۲-۱۵- نحوه‌ی خروجی گرفتن از کد در ترمینال *VS Code*



شکل ۲-۱۶- خروجی برنامه چاپ "Hello World!" در ترمینال *VS Code*

طبق شکل ۲-۱۶ مشاهده می‌کنید که در قسمت ترمینال *VS Code*، کد اجرا شده و "Hello World!" چاپ می‌شود.

فصل سوم

آشنایی مقدماتی با ساختار زبان برنامه‌نویسی پایتون

۳) آشنایی با مقدمات و ساختار زبان برنامه‌نویسی پایتون

تا به اینجا درباره کاربردها، ویژگی‌ها، اهمیت یادگیری زبان پایتون، نصب و اجرای کدهای پایتون در VS Code مورد بحث قرار گرفت. در این فصل به ساختار پایتون از جمله نحوه کامنت‌گذاری، آشنایی بیشتر با تابع چاپ، عمل‌گرها و اولویت آن‌ها در پایتون و... پرداخته خواهد شد.

۳-۱) نحوه کامنت‌گذاری در پایتون

هنگام نوشتن کد با زبان برنامه‌نویسی پایتون، باید به گونه‌ای کدنویسی کرد که دیگران آن کد را درک کنند. اختصاص دادن نام‌های واضح به متغیرها، تعریف توابع کوتاه و سازماندهی کدها همگی راه‌های بسیار عالی برای خوانایی کدها می‌باشد. روش دیگری برای افزایش خوانایی کدها، نوشتن کامنت یا کامنت‌گذاری^{۲۶} در پایتون است. وقتی یک کد را کامنت کنید، هنگام اجرا کردن آن کد دیگر اجرا نمی‌شود و پایتون آن کد را به عنوان کامنت در نظر گرفته و در ترمینال نمایش داده نمی‌شود.

دو نوع کامنت‌گذاری در زبان برنامه‌نویسی پایتون وجود دارد:

- کامنت‌گذاری تکخطی^{۲۷}

- کامنت‌گذاری چندخطی^{۲۸}

معمولًا برای کامنت‌گذاری تک خطی از علامت هشتگ (#) استفاده می‌شود و برای کامنت‌گذاری چند خطی از علامت "##" استفاده می‌شود. زبان برنامه‌نویسی پایتون هر چیزی که بعد از نماد هشتگ باید را تا آخر خط نادیده می‌گیرد.

²⁶ Comment

²⁷ Single-Line Comment

²⁸ Multiple Line Comment

نکته: اگر قبل هشتگ کد نوشته شود، پایتون آن کد را اجرا خواهد کرد.

python example - Introduction.py

```
1 # Comment in Python
2
3 # ** Single Line Comment **
4 # print("Hello World")
5
6 # ** Multiple Line Comment **
7 """
8 کامنت چند خطی
9 Lorem ipsum dolor sit amet, consectetur adipiscing elit,
10 sed do eiusmod tempor incididunt ut labore et dolore magna aliqua.
11 Ut enim ad minim veniam,
12 quis nostrud exercitation ullamco laboris nisi ut aliquip ex ea commodo consequat.
13 Duis aute irure dolor in reprehenderit in voluptate velit esse.
14 Excepteur sint occaecat cupidatat non proident,
15 sunt in culpa qui officia deserunt mollit anim id est laborum.
16 سلام
17 به امید دیدار
18 """
```

شکل ۳-۱- نحوه کامنت‌گذاری تک‌خطی و چندخطی در پایتون

مزایای کامنت‌گذاری:

- افزایش خوانایی کد
- جلوگیری از اجرا در هنگام تست کد
- قابل درک بودن کد برای برنامه‌نویسان دیگر
- داشتن کد تمیز

(print) تابع چاپ آشنایی ۲-۳

دستور print در پایتون برای چاپ مقادیر، پیام‌ها، نتایج محاسباتی و اطلاعاتی که کاربر خواسته، در ترمینال استفاده می‌شود.

تابع print پارامترهای مختلفی دارد که هر کدام مقادیر مشخصی می‌گیرند. مهم‌ترین پارامترها در جدول ۱-۳ آمده است.

جدول ۱-۳ - پارامترهای مربوط به تابع print

پارامتر	توضیحات
Object(s)	هر چیزی که می‌خواهد در خروجی چاپ کنید (رشته‌ها، اعداد، ...).
sep=""	این پارامتر اختیاری است و نبود آن هیچ مشکلی ایجاد نخواهد کرد. اگر بیش از یک خروجی داشته باشد، باتوجه به مقداری که به این پارامتر داده می‌شود، این خروجی‌ها را جداسازی می‌کند. مقدار پیش‌فرض: ""
end=""	توجه داشته باشید که مقداری که می‌خواهید بدهید باید داخل تک کوتيشن (' ') یا دابل کوتيشن (" ") باشد.
	این پارامتر اختیاری است و نبود آن هیچ مشکلی ایجاد نخواهد کرد. مقدار این پارامتر، همانی است که در آخر خروجی چاپ می‌شود. مقدار پیش‌فرض این آرگومان، کاراکتر خط جدید(n) است.

python example - Introduction.py

```
22 # تابع درونی print()
23 # print(Object, sep="separator", end="end")
24 # sep:
25 print("Hello", "World", 4 , 25, sep="-") # (-)
26 print("Hello", "World", 4 , 25, sep="*") # (*)
27 print("Hello", "World", 4 , 25, sep="#") # (#)
28 print("Hello", "World", 4 , 25, sep=" ") # ( )
```

شکل ۳-۲- مثالی از پارامتر *sep* در تابع *print*

خروجی شکل ۳-۲:

Hello-World-4-25

Hello*World*4*25

Hello#World#4#25

Hello World 4 25

python example - Introduction.py

```
40 # تابع درونی print()
41 # print(Object, sep="separator", end="end")
42 # end: Default value of end is: \n (New Line)
43
44 print("First Example ", end="***") # => New value of end is: "***"
45 print("Second Example ", 4 , 25, end="#") # => New value of end is: "#"
```

شکل ۳-۳- مثالی از پارامتر *end* در تابع *print*

*First Example ***Second Example 4 25#*

۳-۳ آشنایی با **f-string** در پایتون

یکی از روش‌های قدرتمند و ساده برای قالب‌بندی رشته‌ها در پایتون Formatted String Literals یا f-strings است. این ویژگی در نسخه Python 3.6 معرفی شد و با استفاده از آن می‌توان مقادیر متغیرها یا خروجی عبارات را مستقیماً در داخل یک رشته قرار داد.

۱-۳-۳) نحوه تعریف **f-strings**

برای استفاده از f-strings

۱. رشته مورد نظر را با پیشوند f یا F تعریف کنید.
۲. مقادیر متغیرها یا عبارات را داخل {} قرار دهید.

۲-۳-۳) ویژگی‌های کلیدی **f-strings**

۱. جای‌گذاری متغیرها: به سادگی می‌توانید مقادیر متغیرها را در رشته قرار دهید.

value = 42

print(f"The value is {value}.")

۲. عبارات درون f-strings: هر عبارت پایتونی را می‌توان مستقیماً درون {} استفاده کرد.

a, b = 5, 3

```
print(f"The sum of {a} and {b} is {a + b}.")
```

۳. قالب‌بندی مقادیر عددی: می‌توانید مقادیر عددی را قالب‌بندی کنید.

pi = 3.14159

```
print(f'Pi to 3 decimal places: {pi:.3f}.')
```

۴. نمایش مقادیر خام: اگر می‌خواهید دقیقاً مقدار داخل {} را ببینید، از **r!=** استفاده کنید:

data = "example"

```
print(f'Raw representation: {data!r}.')
```

۵. استفاده از توابع درون f-strings: می‌توانید مستقیماً تابع را فراخوانی کنید:

name = "world"

```
print(f'Hello, {name.upper()}!')
```

۳-۳-۳) مزایای f-strings

۱. از روش‌های قدیمی‌تر قالب‌بندی (مانند `%` یا `format()`) سریع‌تر هستند.
۲. خوانایی و نوشتتن کد را آسان‌تر می‌کنند.
۳. از قابلیت‌های قدرتمندی مانند جایگذاری متغیر، عبارات و حتی قالب‌بندی اعداد به صورت دلخواه پشتیبانی می‌کنند.
۴. با انواع مختلف داده‌ها، از جمله رشته‌ها، اعداد، فهرست‌ها و دیکشنری‌ها می‌توان کار کرد.
۵. f-strings معمولاً کد را کوتاه‌تر و فشرده‌تر می‌کنند.
۶. کاهش تعداد خطاهای

نکات:

- ✓ نیاز به نسخه Python 3.6 یا جدیدتر: در نسخه‌های قدیمی‌تر پایتون، f-strings پشتیبانی نمی‌شوند.
- ✓ استفاده از `f` یا `F`: پیشوند رشته می‌تواند کوچک یا بزرگ باشد.
- ✓ خواندن و نگهداری راحت‌تر کد: به دلیل ادغام مستقیم متغیرها در متن، کد واضح‌تر است.

۴-۳-۳) مثال مربوط به print در تابع f-string

در ادامه به شکل از f-string توجه کنید.

```
55 # ** F-Strings in Python **  
56  
57 name = "Reza"  
58 age = 23  
59 score = 18.23  
60  
61 print(f"My name is {name} and I am {age} years old.")  
62 print(f"I got a grade of {score} in math.")
```

شکل ۴-۳-۴- مثالی از *print f-string* در تابع

خروجی شکل ۴-۳:

My name is Reza and I am 23 years old.

I got a grade of 18.23 in math.

۴-۳) انواع عملگرها در پایتون

عملگرها^{۲۹} برای انجام عملیات روی متغیرها و داده‌ها استفاده می‌شوند. عملگر نمادی است که بر اساس یک تعریف، عملیات خاصی را بین دو عملوند(متغیر یا داده) انجام می‌دهد. عملگرها انواع مختلفی دارند که در ادامه به هر کدام از آن‌ها پرداخته خواهد شد.

²⁹ Operators

انواع عملگرها:

- عملگرهای ریاضی(حسابی)^{۳۰}
- عملگرهای مقایسه‌ای^{۳۱}
- عملگرهای تخصیص(انتسابی)^{۳۲}
- عملگرهای منطقی^{۳۳}

۱-۴-۳) عملگرهای ریاضی

عملگرهای ریاضی در پایتون برای انجام محاسبات مختلف بر روی اعداد استفاده می‌شوند. این عملگرها به شما امکان می‌دهند عملیات جمع، تفریق، ضرب، تقسیم، توان و ... را بر روی اعداد صحیح و اعشاری انجام دهید. در ادامه به انواع عملگرهای ریاضی در جدول ۳-۲ اشاره خواهد شد.

جدول ۳-۲- عملگرهای ریاضی در پایتون به همراه مثال

نوع عملگر	توضیحات	مثال
جمع (+)	برای جمع کردن دو یا چند عدد استفاده می‌شود.	Result = 20 + 75 print(result) # Output: 95
تفریق (-)	برای کم کردن یک عدد از عدد دیگر استفاده می‌شود.	result = 15 – 8 print(result) # Output: 7

^{۳۰} Arithmetic Operators

^{۳۱} Comparison Operators

^{۳۲} Assignment Operators

^{۳۳} Logical Operators

Result = 5 * 20	برای ضرب کردن دو یا چند عدد استفاده می‌شود.	ضرب (*)
print(result)		
# Output: 100		
Result = 50 / 2	برای تقسیم کردن یک عدد بر عدد دیگر استفاده می‌شود.	تقسیم (/)
print(result)		
# Output: 25		
Result = 17 // 4	برای تقسیم کردن دو عدد و نادیده گرفتن باقی‌مانده استفاده می‌شود(خارج قسمت تقسیم).	تقسیم صحیح (//)
print(result)		
# Output: 4		
result = 2 ** 3	به توان رساندن یک عدد	توان (**)
print(result)		
# Output: 8		
Result = 17 % 4	برای بدست آوردن باقی‌مانده تقسیم دو عدد استفاده می‌شود.	باقی‌مانده (%)
print(result)		
# Output: 1		

۲-۴-۳) عملگرهای مقایسه‌ای

عملگرهای مقایسه‌ای در پایتون برای مقایسه دو مقدار و تعیین این‌که آیا آن‌ها با یکدیگر برابر، بزرگ‌تر یا کوچک‌تر هستند، استفاده می‌شوند. معمولاً خروجی این نوع عملگرها به صورت بولین (درست یا نادرست) می‌باشد. این عملگرها در عبارات شرطی و ساختارهای کنترلی مانند if و while کاربرد دارند. در ادامه به انواع عملگرهای مقایسه‌ای در جدول ۳-۳ اشاره خواهد شد.

جدول ۳-۳- انواع عملگرهای مقایسه‌ای در پایتون به همراه مثال

نوع عملگر	توضیحات	مثال

<pre>x = 10</pre>		
<pre>y = 15</pre>	برای بررسی اینکه آیا دو مقدار با یکدیگر برابر هستند یا خیر، استفاده	مساوی
<pre>result = x == y</pre>	می‌شود.	(==)
<pre>print(result)</pre>		
<pre># Output: False</pre>		
<pre>x = 45</pre>		
<pre>y = 40</pre>	برای بررسی اینکه آیا دو مقدار با یکدیگر برابر نیستند یا خیر، استفاده	غیرمساوی
<pre>result = x != y</pre>	می‌شود.	(!=)
<pre>print(result)</pre>		
<pre># Output: True</pre>		
<pre>x = 20</pre>		
<pre>y = 10</pre>	برای بررسی اینکه آیا یک مقدار از مقدار دیگر بزرگتر است یا خیر،	بزرگتر از
<pre>result = x > y</pre>	استفاده می‌شود.	(>)
<pre>print(result)</pre>		
<pre># Output: True</pre>		
<pre>x = 45</pre>		
<pre>y = 2</pre>	برای بررسی اینکه آیا یک مقدار از مقدار دیگر کوچکتر است یا خیر،	کوچکتر از
<pre>result = x < y</pre>	استفاده می‌شود.	(<)
<pre>print(result)</pre>		
<pre># Output: False</pre>		
<pre>x = 56</pre>		
<pre>y = 79</pre>	برای بررسی اینکه آیا یک مقدار از مقدار دیگر بزرگتر است یا با آن برابر	بزرگتر یا مساوی
<pre>result = x >= y</pre>	است، استفاده می‌شود.	(>=)
<pre>print(result)</pre>		
<pre># Output: False</pre>		
<pre>x = 9</pre>		
<pre>y = 79</pre>	برای بررسی اینکه آیا یک مقدار از مقدار دیگر کوچکتر است یا با آن برابر	کوچکتر یا مساوی
<pre>result = x <= y</pre>	است، استفاده می‌شود.	(<=)
<pre>print(result)</pre>		
<pre># Output: True</pre>		

۳-۴-۳ عملگرهای تخصیص

عملگرهای تخصیص در پایتون برای انتساب مقادیر به متغیرها استفاده می‌شوند. این عملگرها نقش اساسی در برنامه‌نویسی پایتون دارند و برای ذخیره داده‌ها در متغیرها و استفاده از آن‌ها در محاسبات و عملیات مختلف کاربرد دارند. برای درک بهتر، در جدول ۴-۳ به عملگرهای تخصیص اشاره شده است.

جدول ۴-۳- انواع عملگرهای تخصیص در پایتون به همراه مثال

نوع عملگر	توضیحات مثال	مثال
عملگر تخصیص ساده (=)	مقدار age برابر ۲۵ می‌باشد و همین مقدار را در خروجی چاپ کرده است.	age = 25 print(age) # Output: 25
عملگر تخصیص با جمع (+)	در خط اول مقدار age برابر ۲۵ می‌باشد، در خط دوم مقدار age را با ۵ جمع کرده و در خود متغیر age قرار داده و در خط سوم print(age) را چاپ کرده است و خروجی برابر ۳۰ شده است.	age = 25 age += 5 print(age) # Output: 30
عملگر تخصیص با ضرب (*)	در خط اول مقدار age برابر ۲۵ می‌باشد، در خط دوم مقدار age را با ۵ تفريح کرده و در متغیر age قرار داده و در خط سوم print(age) را چاپ کرده است و خروجی برابر ۲۰ شده است.	age = 25 age *= 5 print(age) # Output: 20
عملگر تخصیص با تقسیم (/=)	در خط اول مقدار age برابر ۲۵ می‌باشد، در خط دوم مقدار age را به ۵ تقسیم کرده و در متغیر age قرار داده و در خط سوم print(age) را چاپ کرده است و خروجی برابر ۵ شده است.	age = 25 age /= 5 print(age) # Output: 5
عملگر تخصیص با باقیمانده (٪)	در خط اول مقدار age برابر ۲۵ می‌باشد، در خط دوم باقیمانده تقسیم age بر ۵ در متغیر age قرار داده و در خط سوم print(age) چاپ کرده است و خروجی برابر ۰ شده است. چون باقیمانده # Output: 0 ۲۵/۵ برابر صفر است.	age = 25 age %= 5 print(age) # Output: 0

age = 5	در خط اول مقدار age برابر ۵ می‌باشد، در خط دوم مقدار age	عملگر تخصیص با توان
age **= 3	را به توان ۳ رسانده و حاصل را در age قرارداده و در خط سوم	(**=)
print(age)	را چاپ کرده است و خروجی برابر ۱۲۵ شده است.	
# Output: 125		

age = 4	در خط اول مقدار age برابر ۴ می‌باشد، در خط دوم خارج	عملگر تخصیص با
age //= 3	قسمت تقسیم age بر ۳ در متغیر age قرار داده و در خط سوم	تقسیم صحیح (//=)
print(age)	را چاپ کرده است و خروجی برابر ۸ شده است.	
# Output: 8		

۴-۴-۳ عملگرهای منطقی

عملگرهای منطقی در پایتون برای ترکیب عبارات شرطی و ایجاد عبارات پیچیده‌تر استفاده می‌شوند. این عملگرهای به شما امکان می‌دهند تا شرایط مختلف را در برنامه خود بررسی کنید و براساس آن‌ها اقدامات لازم را انجام دهید. در ادامه به بررسی انواع مختلف از عملگرهای منطقی در جدول ۳-۵ اشاره می‌شود.

جدول ۳-۵- انواع عملگرهای منطقی در پایتون به همراه مثال

نوع عملگر	توضیحات	مثال
and (و)	برای بررسی اینکه آیا هر دو عبارت درست هستند یا خیر، استفاده می‌شود. اگر هر دو عبارت درست باشد، True برمی‌گرداند.	Age = 20 Car_Certificate = True if age >= 18 and Car_Certificate: print("Yo can Buy Car")
or (یا)	برای بررسی اینکه آیا حداقل یکی از عبارات درست است یا خیر، استفاده می‌شود. اگر بین دو عبارت، یکی از عبارات یا هر دو عبارت درست باشد، True بر می‌گرداند.	Is_sunny = True is_weekend = False if is_sunny or is_weekend: print("You can go Cinema.")

```

Is_raining = False
if not is_raining:
    print("Today is Sunny.")

```

برای ضد کردن نتیجه یک عبارت استفاده می‌شود. not (نیست)

دو عبارت A و B را در نظر بگیرید:

برای عملگر منطقی *and* •

جدول ۳-۶- مقایسه‌ی دو شرط با عملگر منطقی *and*

نتیجه	B	A
درست(True)	درست(True)	درست(True)
نادرست(False)	نادرست(False)	درست(True)
نادرست(False)	درست(True)	نادرست(False)
نادرست(False)	نادرست(False)	نادرست(False)

python example - Introduction.py

```

71 # Logical Operator (and)
72
73 print(True and True)
74 print(True and False)
75 print(False and True)
76 print(False and False)
77

```

شکل ۳-۵- مثالی از عملگر منطقی *and* در پایتون

خروجی شکل ۳-۵:

True

False

False

False

برای عملگر منطقی or •

جدول ۳-۷- مقایسه‌ی دو شرط با عملگر منطقی *or*

نتیجه	B	A
درست (True)	درست (True)	درست (True)
درست (True)	نادرست (False)	درست (True)
درست (True)	درست (True)	نادرست (False)
نادرست (False)	نادرست (False)	نادرست (False)

python example - Introduction.py

```
86 # Logical Operator (or)
87
88 print(True or True)
89 print(True or False)
90 print(False or True)
91 print(False or False)
```

شکل ۳-۶- مثالی از عملگر منطقی *or* در پایتون

خروجی شکل ۳-۶:

True

True

True

False

۴-۵) اولویت عملگرها

اولویتبندی عملگرها در پایتون تعیین می‌کند که در یک عبارت پیچیده، کدام عملگرها ابتدا اجرا می‌شوند. پایتون

از قوانینی مشابه ریاضیات برای تعیین اولویت عملگرها استفاده می‌کند. در ادامه، عملگرهای پایتون بر اساس

اولویت، از بالاترین به پایین‌ترین، دسته‌بندی شده‌اند:

جدول ۳-۸- اولویتبندی عملگرها در پایتون

اولویت	عمل‌گرها	عمل‌گرها	عمل‌گرها
۱	توان (**)	به توان رساندن یک عدد با عددی دیگر	
۲	ضرب (*) و تقسیم (/)	ضرب و تقسیم از چپ به راست انجام می‌شوند	
۳	جمع (+) و تفریق (-)	جمع و تفریق از چپ به راست انجام می‌شوند	
۴	مقایسه	مقایسه دو مقدار با یکدیگر $(==, !=, <, >, <=, >=)$	
۵	منطقی	نسبت به or و and اولویت دارد. نسبت به not نسبت به or و and اولویت دارد.	(not, and, or)
۶	تخصیص(انتسابی)	انتساب مقادیر به متغیرها $(//=, /=, =*, -=, =+, =)$	

۳-۵) معرفی توابع وابسته در پایتون

در برنامه‌نویسی شیء‌گرا، اشیاء شامل داده‌ها و رفتارهایی هستند که این داده‌ها را مدیریت می‌کنند. این رفتارها در قالب توابع وابسته یا همان متدها^{۳۴} تعریف می‌شوند. توابع وابسته، توابعی هستند که به یک کلاس یا شیء خاص تعلق دارند و برای دسترسی یا تغییر وضعیت داخلی شیء استفاده می‌شوند. این توابع ارتباط نزدیکی با داده‌های مرتبط با شیء دارند و معمولاً به عنوان ابزاری برای تعریف رفتار و قابلیت‌های اشیاء به کار می‌روند.

توابع وابسته به عنوان اجزای کلیدی در زبان‌های شیء‌گرا، از جمله پایتون، شناخته می‌شوند. این توابع معمولاً درون یک کلاس تعریف می‌شوند و برای مدیریت و انجام عملیات روی داده‌های مرتبط با یک شیء خاص استفاده می‌شوند. هنگام فراخوانی یک تابع وابسته، شیء مربوطه به صورت خودکار به تابع منتقل می‌شود، که این فرآیند امکان دسترسی به داده‌ها و ویژگی‌های داخلی آن شیء را فراهم می‌کند. در بخش‌های آینده در مورد برنامه‌نویسی شیء‌گرا پرداخته خواهد شد.

³⁴ Methods

فصل چهارم

متغیرها و انواع داده‌ها در زبان برنامه‌نویسی پایتون

۴) متغیرها و انواع داده‌ها در زبان برنامه‌نویسی پایتون

۱-۴) متغیرها در پایتون

متغیرها^{۳۵} فضاهایی از حافظه برای ذخیره‌سازی داده‌ها هستند. برای درک آسان متغیرها، تصور کنید سبدی دارید که میوه و سبزیجات را در آن قرار داده‌اید، حال اون سبد همان متغیر است و میوه و سبزیجات همان داده‌هاست. برای تعریف یک متغیر در پایتون، از علامت مساوی (=) استفاده می‌شود. به این ترتیب، نام متغیر را در سمت چپ و مقدار متغیر را بعد از علامت مساوی، در سمت راست قرار داده می‌گیرد.

Custom_variable_name = value

نام متغیر (نام دلخواه) می‌باشد (نقش سبد میوه و سبزیجات). ♦

value مقداری است که متغیر می‌گیرد (نقش میوه و سبزیجات). ♦

قوانين نام‌گذاری متغیرها در پایتون:

- نام متغیر باید با حروف یا زیرخط (_) شروع شود. اما نمی‌تواند با عدد شروع شود. •
- نام متغیر به حروف کوچک و بزرگ حساس است. به عنوان مثال، name با Name متفاوت است. •
- برای نام‌گذاری متغیرها از نام‌های گویا و توصیفی استفاده کنید تا خوانایی کد شما افزایش یابد. •
- می‌توانید چندین متغیر را در یک خط تعریف کنید. •

متغیرها می‌توانند انواع مختلفی از داده‌ها را ذخیره کنند. در جدول ۱-۴ انواع مختلفی از انواع داده‌ها در پایتون آمده است.

³⁵ Variables

جدول ۴-۱- انواع داده‌ها در پایتون

نوع داده	توضیحات	مثال
داده‌های عددی صحیح (int)	برای ذخیره اعداد بدون اعشار	50, 20, 23
داده‌های عددی اعشاری (float)	برای ذخیره اعداد با اعشار	40.55, 23.64, 33.2, 26.0001
داده‌های رشته‌ای (str)	برای ذخیره متن	"Hello World", "سلام", "ایران"
تاپل‌ها (tuple)	برای ذخیره مجموعه‌ای از مقادیر از انواع مختلف (امکان تغییر مقادیر داده شده بعد از تعریف وجود ندارد)	names = ("Jason", "Rose", "Jack")
فهرست (list)	برای ذخیره مجموعه‌ای از مقادیر از انواع مختلف (امکان تغییر مقادیر داده شده بعد از تعریف وجود دارد)	cars = ["BMW", "Benz", ...]
آرایه‌ها (array)	برای ذخیره مجموعه‌ای از مقادیر	names = ["John", "Mary", "Peter"]
مجموعه (set)	برای ذخیره مجموعه‌ای از مقادیر منحصر به فرد	fruits = {"apple", "banana"}
دیکشنری (dictionary)	برای ذخیره مجموعه‌ای از جفت کلید-مقدار	student_data = { "id": 123, "name": "John Doe", "courses": ["Math", "Science"]}
بولین (Boolean)	برای ذخیره مقادیر True یا False	x = True y = False

نکته: برای این‌که نوع داده‌ی یک متغیر را تشخیص دهید، از دستور زیر استفاده می‌شود:

print(type(variable Name))

```
1 # Type function
2
3 name = "Reza"
4 age = 23
5 score = 18.23
6 favorite_colors = ["yellow", "red", 'blue']
7
8 print(f"Type of name:{name} is {type(name)}")
9 print(f"Type of age:{age} is {type(age)}")
10 print(f"Type of score:{score} is {type(score)}")
11 print(f"Type of favorite_colors:{favorite_colors} is {type(favorite_colors)}")
```

شكل ۴-۱- مثالی از دستور *type* در پایتون

خروجی شکل ۱-۴:

Type of name:Reza is <class 'str'>

Type of age:23 is <class 'int'>

Type of score:18.23 is <class 'float'>

Type of favorite_colors:['yellow', 'red', 'blue'] is <class 'list'>

۴-۲) انواع داده‌ها^{۳۶} در پایتون

۴-۲-۱) اعداد در پایتون

در ساختار پایتون می‌توان سه نوع عدد را تعریف کرد:

^{۳۶} Data Types

- int
- float
- complex

۱-۲-۴) نوع عددی int در پایتون

نوع عددی int همان اعداد صحیح در ریاضیات است. نوع عددی int می‌تواند یک عدد صحیح، مثبت یا منفی، بدون اعشار، با طول نامحدود باشد. مقداری که با نوع عددی int تعریف می‌شود، می‌تواند هر طولی داشته باشد.

- first_int_number = 25
- second_int_number = -763
- third_int_number = 0
- fourth_int_number = 9852145215615

۲-۱-۲-۴) نوع عددی float در پایتون

نوع عددی float همان اعداد اعشاری در ریاضیات است. نوع عددی float یک عدد مثبت یا منفی است که شامل یک یا چند اعشار است و می‌تواند تا ۱۵ رقم اعشار دقیق باشد.

- first_float_number = 2.96
- second_float_number = -653.123
- third_float_number = 0.23

۳-۱-۲-۴) نوع عددی complex در پایتون

نوع عددی complex همان اعداد مخلوط در ریاضیات است. این نوع اعداد یک قسمت واقعی و یک قسمت موهومی دارند که در پایتون با حرف *j* نشان داده می‌شوند.

- first_complex_number = 3+5j
- second_complex_number = 5j
- third_complex_number = -5j

python example - Numbers.py

```
21 # اعداد در پایتون (Numbers in Python)
22 # int numbers
23
24 int_number_1 = 256
25 int_number_2 = -985
26 int_number_3 = 0
27
28 print(f"Type of {int_number_1} is {type(int_number_1)}")
29 print(f"Type of {int_number_2} is {type(int_number_2)}")
30 print(f"Type of {int_number_3} is {type(int_number_3)}")
```

شکل ۴-۲- مثالی از نوع عددی *int* در پایتون

خروجی شکل ۴-۲:

Type of 256 is <class 'int'>

Type of -985 is <class 'int'>

Type of 0 is <class 'int'>

python example - Numbers.py

```
39 # اعداد در پایتون (Numbers in Python)
40     # float numbers
41
42 float_number_1 = 10.268
43 float_number_2 = -563.598436
44 float_number_3 = 0.456
45 float_number_4 = -0.5236982
46
47 print(f"Type of {float_number_1} is {type(float_number_1)}")
48 print(f"Type of {float_number_2} is {type(float_number_2)}")
49 print(f"Type of {float_number_3} is {type(float_number_3)}")
50 print(f"Type of {float_number_4} is {type(float_number_4)}")
```

شکل ۴-۳- مثالی از نوع عددی *float* در پایتون

خروجی شکل ۴-۳:

Type of 10.268 is <class 'float'>

Type of -563.598436 is <class 'float'>

Type of 0.456 is <class 'float'>

Type of -0.5236982 is <class 'float'>

```
60 # Numbers in Python (اعداد در پایتون)
61     # complex numbers
62
63 complex_number_1 = 10 + 5j
64 complex_number_2 = -25j
65 complex_number_3 = 1j
66
67 print(f"Type of {complex_number_1} is {type(complex_number_1)}")
68 print(f"Type of {complex_number_2} is {type(complex_number_2)}")
69 print(f"Type of {complex_number_3} is {type(complex_number_3)}")
```

شکل ۴-۴- مثالی از نوع عددی *complex* در پایتون

خروجی شکل ۴-۴:

Type of (10+5j) is <class 'complex'>

Type of (-0-25j) is <class 'complex'>

Type of 1j is <class 'complex'>

۲-۲-۴) رشته‌ها در پایتون

در پایتون، نوع داده رشته^{۳۷}، یک توالی از کاراکترها است به عبارت دیگر به تعدادی کاراکتر گفته می‌شود که داخل یک جفت کوئیشن('') یا دابل کوئیشن("") قرار بگیرند.

³⁷ String

نکته: در قسمت کامنت‌گذاری گفته شد که برای کامنت چند خطی از " " استفاده می‌شود. اما اگر " " به متغیری تعلق بگیرد در نقش رشته است و اگر به متغیری تعلق نگیرد در نقش کامنت می‌باشد.

python example - strings.py

```
1 # String in Python (رشته‌ها در پایتون)
2
3 # String with single-quotation
4 single_quotation_string = 'In the name of GOD'
5 print(f"type of single_quotation_string is {type(single_quotation_string)}")
6
7 # String with double-quotation
8 double_quotation_string = "Hello, World!"
9 print(f"type of double_quotation_string is {type(double_quotation_string)}")
10
11 # String with triple-quotation
12 triple_quotation_string = """
13 In the name of GOD
14 Hello, World!
15 How are you?
16 """
17 print(f"type of triple_quotation_string is {type(triple_quotation_string)}")
```

شكل ۴-۵- تعریف رشته‌ها در پایتون

خروجی شکل ۴-۵:

type of single_quotation_string is <class 'str'>

type of double_quotation_string is <class 'str'>

type of triple_quotation_string is <class 'str'>

۴-۲-۲-۱) توابع وابسته مربوط به رشته‌ها در پایتون

جدول ۴-۲- جدول مربوط به توابع وابسته موجود در نوع داده‌ی رشته‌ای

نوع توابع وابسته	توضیحات
capitalize()	اولین حروف کلمات را با حروف بزرگ بازنویسی می‌کند.
upper()	کل کاراکترهای یک رشته را به حروف بزرگ تبدیل می‌کند.
lower()	کل کاراکترهای یک رشته را به حروف کوچک تبدیل می‌کند.
title()	تبدیل کاراکتر اول هر کلمه در یک رشته به حروف بزرگ
casefold()	کلمات را با حروف کوچک بازنویسی می‌کند.
swapcase()	تبدیل حروف کوچک به بزرگ و بالعکس.
center(length, character)	بازوجه به مقدار length که می‌گیرد، رشته مورد نظر را در مرکز قرار می‌دهد، با استفاده از character مشخص شده (به صورت پیش فرض فضای خالی (space) است) فضا را پر می‌کند.
count(value, start, end)	با توجه به value که می‌گیرد، آن را در یک رشته بررسی می‌کند و تعداد دفعاتی که تکرار شده را برمی‌گرداند. مقادیر start و end اختیاری هستند. یک عدد صحیح است که جستجو را از آن موقعیت شروع می‌کند و مقدار start پیش فرض برابر ۰ است.

یک عدد صحیح است که در آن موقعیت جستجو را پایان می‌دهد. پیش فرض انتهای رشته است.

در این تابع وابسته، با استفاده از مقدار tabspace که یک عدد صحیح می‌باشد تعداد فاصله هر tab در رشته مشخص می‌شود. یعنی در هر tab چه تعداد فضای خالی ایجاد می‌شود (پیش فرض مقدار آن ۸ است).

رشته را برای یک مقدار value جستجو می‌کند و موقعیت جایی که value پیدا شده است را برمی‌گرداند.

find(value, start, end)

جستجو را از کجا شروع شود. مقدار پیش فرض برابر ۰ است.

جستجو کجا پایان یابد. پیش فرض انتهای رشته است.

رشته را با توجه به مقدار value جستجو می‌کند و آخرین موقعیت جایی که پیدا کرده را برمی‌گرداند. با توجه به مقداری که می‌گیرد، اگر آن مقدار در یک رشته تکرار شده باشد، تابع وابسته find() اولین مقداری که از سمت چپ به راست پیدا می‌کند، موقعیت آن را برمی‌گرداند اما تابع وابسته rfind() آخرین مقداری که از سمت چپ به راست پیدا می‌کند، موقعیت آن را برمی‌گرداند.

rfind(value, start, end)

جستجو را از کجا شروع شود. مقدار پیش فرض برابر ۰ است.

جستجو کجا پایان یابد. پیش فرض انتهای رشته است.

این تابع وابسته مقدارهای مشخص شده را قالب بندي می‌کند و آنها را دورن نگهدارنده‌های متغیر مربوطه قرار می‌دهد.

format(value1, value2...)

با توجه به مقدار value که می‌گیرد، آن value را در رشته جستجو می‌کند و اگر پیدا کند موقعیت آن value در رشته را برمی‌گرداند.

index(value, start, end)

جستجو را از کجا شروع شود. مقدار پیش فرض برابر ۰ است.

جستجو کجا پایان یابد. پیش فرض انتهای رشته است.

رشته را برای یک مقدار مشخص جستجو می کند و آخرین موقعیت جایی که پیدا شده را برمی گردد. با توجه به مقداری که می گیرد، اگر آن مقدار در یک رشته تکرار شده باشد، تابع وابسته `index()` اولین مقداری که از سمت چپ به راست پیدا می کند، موقعیت آن را برمی گردد اما تابع وابسته `rindex()` آخرین مقداری که از سمت چپ به راست پیدا می کند، موقعیت آن را برمی گردد. تفاوتی که `rindex` با `index` دارد این است که اگر مقدار در رشته وجود نداشته باشد، `rfind` مقدار ۱- را برمی گردد اما `rindex` ارور می دهد.

`rindex(value, start, end)`

جستجو را از کجا شروع شود. مقدار پیش فرض برابر ۰ است.
end: جستجو کجا پایان یابد. پیش فرض انتهای رشته است.

اگر همه کاراکترهای رشته به صورت عددی و الفبایی باشند، `True` را برمی گردد.

اگر در رشته ها حروف (space, !, #, %, &, ?, ...) باشد مقدار `False` را برمی گردد.

`isalnum()`

اگر همه کاراکترهای رشته به صورت کاراکتر الفبا باشند، `True` را برمی گردد.

`isalpha()`

اگر همه کاراکترهای رشته کاراکترهای `ascii` باشند، `True` را برمی گردد.

`isascii()`

اگر همه کاراکترهای رشته به صورت عددی و رقم باشند، `True` را برمی گردد.

`isdigit()`

اگر تمام کاراکترهای رشته با حروف کوچک باشند، `True` را برمی گردد.

`islower()`

اگر همه کاراکترهای رشته به صورت عددی و رقم باشند، `True` را برمی گردد.

تفاوت اصلی بین دو تابع `isnumeric()` و `isdigit()` این است: تابع وابسته `isnumeric()` اعداد رومی را هم به عنوان عدد و رقم می شناسد.

`isnumeric()`

اگر همه کاراکترهای رشته space یا فاصله باشند، True را برمی‌گرداند.	isspace()
اگر فقط یک کاراکتر space نباشد، مقدار False برمی‌گرداند.	
اگر رشته از قوانین title پیروی کند، True را برمی‌گرداند.	istitle()
اگر همه کلمات در یک متن با حروف بزرگ شروع شوند، و بقیه کلمات حروف کوچک هستند، در غیر این صورت False	
اگر همه کاراکترهای رشته با حروف بزرگ باشند، True را برمی‌گرداند	isupper()
یک مقدار iterable مثل فهرست یا توپل یا ... می‌گیرد و به یک رشته تبدیل می‌کند.	join(iterable)
باقیه به مقدار characters که می‌گیرد، آن مقدار را در رشته جستجو می‌کند و اگر آن مقدار در رشته باشد، آن را از هر دو سمت راست و چپ حذف می‌کند.	strip(characters)
باقیه به مقدار characters که می‌گیرد، آن مقدار را در رشته جستجو می‌کند و اگر آن مقدار در رشته باشد، آن را از سمت چپ حذف می‌کند.	lstrip(characters)
باقیه به مقدار characters که می‌گیرد، آن مقدار را در رشته جستجو می‌کند و اگر آن مقدار در رشته باشد، آن را از سمت راست حذف می‌کند.	rstrip(characters)
با توجه به مقدار value که می‌گیرد، اگر آن مقدار در داخل رشته باشد، آن رشته را براساس مقدار گرفته شده به سه قسمت تقسیم می‌کند و هر قسمت را به عنوان آیتم توپل در نظر می‌گیرد و در نهایت یک توپل با سه آیتم برمی‌گرداند که آیتم اول شامل قسمت قبل آن مقدار در رشته است مشخص شده است، آیتم دوم شامل همان مقداری که گرفته، و آیتم سوم شامل قسمت بعد آن مقدار در رشته است.	partition(value)
این تابع وابسته برای جایگزینی یک مقدار با مقدار جدید استفاده می‌شود و دو مقدار می‌گیرد که مقدار اول همان oldValue است که می‌خواهد حذف و جایگزین کنید و مقدار دوم newValue است که می‌خواهد جایگزین کنید.	replace(oldValue, newValue, count)

: اختیاری است. بصورت عددی می‌باشد. چند مورد از مقدار قدیمی را count می‌خواهید جایگزین کنید. پیش فرض همه موارد است.

باتوجه به مقدار separator که می‌گیرد، آن مقدار را به عنوان جدا کننده در نظر می‌گیرد و طبق اون، یک رشته را طبق separator جدا می‌کند و یک فهرست برمی‌گردد.
هر دو مقدار ورودی، اختیاری است.

split(separator, maxsplit)

: مشخص می‌کند که چند تقسیم انجام شود. مقدار پیش‌فرض ۱- است که «همه موارد» است.

(white space)“ : مقدار پیش فرض separator

باتوجه به مقداری که می‌گیرد، آن مقدار را به عنوان جدا کننده در نظر می‌گیرد و طبق اون، یک رشته را با توجه به مقدار گرفته شده، جدا می‌کند و یک فهرست برمی‌گردد.

تنها تفاوت بین split() و rsplit() استفاده مقدار maxsplit که به rsplit() داده می‌شود. اگر مقدار maxsplit تنظیم شده باشد،تابع rsplit() یک رشته را از سمت راست جدا می‌کند، در حالی که تابع وابسته split() از سمت چپ جدا می‌کند.

rsplit(separator, maxsplit)

: مشخص می‌کند که چند تقسیم انجام شود. مقدار پیش‌فرض ۱- است که «همه موارد» است.

(white space)“ : مقدار پیش فرض separator

یک رشته را با توجه به شکست خط جدا می‌کند، معمولاً خط با “\n” شکسته می‌شود.

splitlines()

اگر رشته با مقدار value شروع شود مقدار True را برمی‌گرداند.

مقادیر start و end اختیاری هستند.

یک عدد صحیح است که جستجو را از آن موقعیت شروع می‌کند و مقدار

startswith(value, start, end)

پیش فرض برابر ۰ است.

یک عدد صحیح است که در آن موقعیت جستجو را پایان می‌دهد. پیش فرض

انتهای رشته است.

اگر رشته با مقدار value به پایان برسد، مقدار True را برمی‌گرداند.

مقادیر start و end اختیاری هستند.

یک عدد صحیح است که جستجو را از آن موقعیت شروع می‌کند و مقدار

endswith(value, start, end)

پیش فرض برابر ۰ است.

یک عدد صحیح است که در آن موقعیت جستجو را پایان می‌دهد. پیش فرض

انتهای رشته است.

۲-۲-۴ مفهوم concatenation در پایتون

به ترکیب و اتصال دو یا چند رشته در پایتون concatenation گفته می‌شود.

python example - strings.py

```
21 # String in Python (رشته‌ها در پایتون)
22
23 # Concentration in Python
24 # Syntax: string_1 + string_2 + ....
25
26 string_1 = "Hello World!"
27 string_2 = "Nice to meet you."
28 string_3 = "Good Luck:")
29
30 string_sums = string_1 + " " + string_2 + " " + string_3
31 print(f"Concentration Method: {string_sums}")
32
33 # Use F-string for concentration
34 print(f"f-string Method: {string_1} {string_2} {string_3}")
```

شکل ۴-۶- مثالی از مفهوم *concentration* در پایتون

خروجی شکل ۴-۶:

Concentration Method: Hello World! Nice to meet you. Good Luck:)

f-string Method: Hello World! Nice to meet you. Good Luck:)

۳-۲-۲-۴ مفهوم indexing در پایتون

معمولاً برای استخراج و دسترسی به یک کاراکتر از یک رشته از مفهوم indexing استفاده می‌شود.

برای استفاده معمولاً طبق کد زیر باید عمل کرد:

myChar = string[index]

نکته: index ها در پایتون از صفر شروع می‌شود. بنابراین برای دسترسی به اولین آیتم باید مقدار صفر(۰) داده شود. اگر مقدار به صورت عدد منفی داده شود، از آخر به اول رشته حرکت می‌کند.

python example - strings.py

```
38 # String in Python (رشته‌ها در پایتون)
39 # Indexing in Python
40
41 string_sample = "Hello, World!"
42 myChar_1 = string_sample[0]
43 myChar_2 = string_sample[1]
44 myChar_3 = string_sample[2]
45 myChar_4 = string_sample[-1] # => in "World!" extract "!"
46 myChar_5 = string_sample[-2] # => in "World!" extract "d"
47 myChar_6 = string_sample[-13] # => in "World!" extract "H"
48
49 print(f"string_sample[0] of 'Hello, World!' is: {myChar_1}")
50 print(f"string_sample[1] of 'Hello, World!' is: {myChar_2}")
51 print(f"string_sample[2] of 'Hello, World!' is: {myChar_3}")
52 print(f"string_sample[-1] of 'Hello, World!' is: {myChar_4}")
53 print(f"string_sample[-2] of 'Hello, World!' is: {myChar_5}")
54 print(f"string_sample[-13] of 'Hello, World!' is: {myChar_6}")
```

شكل ۴-۷- مثالی از مفهوم *indexing* در پایتون

خروجی شکل ۷-۴

string_sample[0] of 'Hello, World!' is: H

string_sample[1] of 'Hello, World!' is: e

string_sample[2] of 'Hello, World!' is: l

string_sample[-1] of 'Hello, World!' is: !

string_sample[-2] of 'Hello, World!' is: d

string_sample[-13] of 'Hello, World!' is: H

۴-۲-۲-۴) مفهوم slicing در پایتون

در پایتون، slicing (برش) به شما این امکان را می‌دهد که بخشی از یک رشته (یا فهرست، تاپل و ...) را استخراج کنید. این کار با استفاده از سینتکس زیر انجام می‌شود:

mySentence = string[start:end:step]

- start: اندیسی که برش از آن شروع می‌شود (شامل می‌شود). اگر ذکر نشود، از ابتدای رشته شروع می‌شود.
- end: اندیسی که برش در آن پایان می‌باید (شامل نمی‌شود). اگر ذکر نشود، تا انتهای رشته ادامه می‌یابد.
- step: گام برش را تعیین می‌کند. مقدار پیشفرض آن ۱ است. اگر مقدار -۱ را بگیرد، start و end نیز تغییر می‌باید و در این صورت مقدار end باید از مقدار start کمتر یا مساوی باشد و نتیجه به این صورت می‌شود که استخراج شده بصورت معکوس در خروجی چاپ می‌شود.

انواع مختلف برش:

- برش رشته از ابتدا تا یک اندیس مشخصی (برای مثال تا اندیس دهم):
x = string[:10]
- برش رشته از یک اندیس مشخصی تا آخر (برای مثال برش از اندیس سوم):
x = string[3:]
- اندیس گذاری منفی (به شکل ۸-۴ توجه کنید).

نکته: برای معکوس کردن یک رشته کافیه کد زیر نوشته شود:

reverse_string = string[::-1]

python example - strings.py

```
58 # String in Python (رشته‌ها در پایتون)
59 # Slicing in Python
60
61 string_sample_2 = "Hello, World! Nice to meet you."
62
63 Hello_extract = string_sample_2[0:6]
64 print(f'Extract 'Hello' from string_sample_2: {Hello_extract}')
65
66 reverse_string_sample_2 = string_sample_2[::-1]
67 print(f'Reverse string sample_2: {reverse_string_sample_2}')
68
69 Meet_extract_and_reverse = string_sample_2[-6:-10:-1]
70 print(f'Extract 'Meet' from string_sample_2 and reverse: {Meet_extract_and_reverse}')
```

شکل ۴-۸-مثالی از مفهوم *slicing* در پایتون

خروجی شکل ۴-۸:

Extract 'Hello' from string_sample_2: Hello,

Reverse string sample_2: .uoy teem ot eciN !dlroW ,olleH

Extract 'Meet' from string_sample_2 and reverse: teem

۴-۲-۲-۳) توابع وابسته‌ای که برای اصلاحات نگارشی، می‌توان برای رشته‌ها به کار برد

• *capitalizer()* تابع وابسته

این تابع وابسته بیشتر برای زبان انگلیسی کاربرد دارد و با اعمال روی یک رشته باعث می‌شود که حرف اول هر کلمه از رشته به حرف بزرگ تبدیل شود. این تابع وابسته مقداری در داخل پرانتز نمی‌گیرد.

• **تابع وابسته upper()**

این تابع وابسته نیز بیشتر برای زبان انگلیسی کاربرد دارد و باعث می‌شود که تمامی حروف در یک رشته به حروف بزرگ انگلیسی تغییر پیدا کند. این تابع وابسته مقداری در داخل پرانتز نمی‌گیرد.

• **تابع وابسته lower()**

این تابع وابسته نیز بیشتر برای زبان انگلیسی کاربرد دارد و باعث می‌شود که تمامی حروف در یک رشته به حروف کوچک انگلیسی تغییر پیدا کند. این تابع وابسته مقداری در داخل پرانتز نمی‌گیرد.

python example - strings.py

```
74 # String in Python (رشته‌ها در پایتون)
75
76 myString = "hello, and welcome to my world."
77
78 # Capitalize()
79 cap_string = myString.capitalize()
80 print(f"Capitalize of myString is: {cap_string}")
81
82 # Upper()
83 upper_string = myString.upper()
84 print(f"Upper of myString is: {upper_string}")
85
86 # Lower()
87 lower_string = myString.lower()
88 print(f"Lower of myString is: {lower_string}")
```

شكل ۴-۹- مثالی از توابع وابسته *Lower* و *Upper* ،*Capitalize*

Capitalize of myString is: Hello, and welcome to my world.

Upper of myString is: HELLO, AND WELCOME TO MY WORLD.

Lower of myString is: hello, and welcome to my world.

• **تابع وابسته title()**

یک تابع وابسته از کلاس رشته‌ها است که برای قالب‌بندی متون به کار می‌رود. عملکرد این تابع وابسته به گونه‌ای است که حروف ابتدایی هر کلمه در رشته را به حروف بزرگ و سایر حروف آن کلمه را به حروف کوچک تبدیل می‌کند. به بیان ساده‌تر، تابع وابسته‌ی `title` متن را به فرم عنوان‌گونه در می‌آورد؛ یعنی هر کلمه در رشته به صورت خودکار با حرف بزرگ آغاز شده و ادامه آن به حروف کوچک نوشته می‌شود.

• **تابع وابسته casefold()**

این تابع وابسته یکی از توابع وابسته‌ای رشته‌ای است که برای تبدیل تمام حروف یک رشته به حروف کوچک استفاده می‌شود. عملکرد این تابع وابسته مشابه تابع وابسته lower است، اما با قدرت و دقت بیشتری عمل می‌کند. تابع وابسته `casefold` برای مقایسه رشته‌ها در زبان‌هایی که دارای تفاوت‌های طریف در حروف بزرگ و کوچک هستند، بسیار کارآمد است؛ زیرا این تابع وابسته تمام حروف را به کوچک‌ترین شکل ممکن و بدون در نظر گرفتن تفاوت‌های زبانی تبدیل می‌کند.

• **تابع وابسته swapcase()**

این تابع وابسته یکی از توابع وابسته‌ای رشته‌ای است که وظیفه تبدیل حروف بزرگ به کوچک و حروف کوچک به بزرگ را در یک رشته بر عهده دارد. به بیان ساده‌تر، این تابع وابسته حالت حروف را جابجا می‌کند؛ یعنی اگر یک حرف در رشته به صورت بزرگ باشد، آن را به کوچک تبدیل می‌کند و برعکس.

python example - strings.py

```
92 # String in Python (رشته‌ها در پایتون)
93
94 myString = "Hello, and welcome To python Tutorial."
95
96 # Title()
97 title_string = myString.title()
98 print(f"Title method for myString: {title_string}")
99
100 # Casefold()
101 casefold_string = myString.casefold()
102 print(f"Casefold method for myString: {casefold_string}")
103
104 # Swapcase()
105 swapcase_string = myString.swapcase()
106 print(f"Swapcase method for myString: {swapcase_string}")
```

شكل ۴ - مثالی از توابع وابسته‌ی *title* و *casefold* و *swapcase*

خروجی شکل ۴ :

Title method for myString: Hello, And Welcome To Python Tutorial.

Casefold method for myString: hello, and welcome to python tutorial.

Swapcase method for myString: hELLO, AND WELCOME tO PYTHON tUTORIAL.

تابع وابسته strip()

این تابع وابسته برای حذف مقادیر مشخص شده در ابتدا و انتهای یک رشته استفاده می‌شود. به عبارت دیگر این تابع وابسته، با توجه به مقداری که گرفته، آن مقدار را از ابتدا و انتهای یک رشته حذف می‌کند. این تابع وابسته یک مقدار دلخواه در داخل پرانتز می‌گیرد.

python example - strings.py

```
110 # String in Python (رشته‌ها در پایتون)
111 # Strip()
112
113 # Example 1
114 String_example_1 = " apples are delicious fruits that come in many different colors. "
115 new_String_example_1 = String_example_1.strip()
116 print(f"Example 2: Before: {String_example_1}\nAfter: {new_String_example_1}")
117
118 # Example 2
119 String_example_2 = "***apples are delicious fruits that come in many different colors.**"
120 new_String_example_2 = String_example_2.strip("*") #Set Custom Character
121 print(f"Example 2: Before: {String_example_2}\nAfter: {new_String_example_2}")
```

شكل ۴-۱۱-۴- مثالی از تابع وابسته strip

خروجی شکل ۱۱-۴:

Example 2: Before: apples are delicious fruits that come in many different colors.

After: apples are delicious fruits that come in many different colors.

*Example 2: Before: ***apples are delicious fruits that come in many different colors.***

After: apples are delicious fruits that come in many different colors.

تابع وابسته replace()

اگر قصد دارید که یک کلمه از یک رشته را حذف کنید و کلمه‌ی جدیدی را جایگزین آن کنید، می‌توانید از این تابع وابسته برای رشته‌ها استفاده کنید.

تابع وابسته Replace در داخل پرانتز سه مقدار می‌گیرد که مقدار اول مقداری است که قصد دارید حذف کنید و با مقدار دوم جایگزین کنید. مقدار دوم مقدار جدید است که جایگزین مقدار قدیمی می‌شود. ضروری است که مقادیر اول و دوم مقداردهی کنید.

مقدار سوم، مقداری اختیاری است و به صورت عددی صحیح مقداردهی می‌شود. در شکل ۱۲-۴ مشاهده می‌کنید که در متن دو تا کلمه "apples" وجود دارد و باید این مقادیر با کلمه "oranges" جایگزین شوند. اگر در تابع وابسته replace مقدار سوم را مقداردهی نکنید، این تابع وابسته تمام مقادیری که به شکل کلمه "apples" می‌باشد، با کلمه "oranges" جایگزین می‌کند اما اگر مقداردهی شود، این تابع وابسته طبق اون عدد، همان تعداد را جایگزین می‌کند. به عبارت دیگر اگر در یک رشته‌ای ۵ کلمه‌ی "apples" وجود داشته باشد، و مقدار "oranges" را بخواهید جایگزین "apples" کنید، اگر مقدار سوم، مقداردهی نشود، تمام کلماتی که "apples" هستند، حذف و به جای آنها "oranges" جایگزین می‌شود، اما اگر مقداردهی انجام شود، برای مثال مقدار ۳، این تابع وابسته ۳ کلمه که از ابتدای رشته جستجو کرده و به شکل "apples" پیدا کرده را حذف و به جای آنها کلمه "oranges" قرار خواهد داد.

```
125 # String in Python (رشته‌ها در پایتون)  
126 # Replace(old value, new value, count)  
127  
128 newString = """Apple is a sweet and delicious fruit that is very healthy.  
129 Apple is also a good disease-fighter and we can eat it every day."""  
130  
131 newString_1 = newString.replace('Apple', 'Orange')  
132 print(f"replace all 'Apple' to 'Orange':\n{newString_1}", end='\n\n')  
133  
134 newString_2 = newString.replace('Apple', 'Orange', 1)  
135 print(f"replace first 'Apple' to 'Orange':\n{newString_2}")
```

شكل ۱۲-۴ - مثالی از تابع وابسته *replace*

خروجی شکل ۱۲-۴ :

replace all 'Apple' to 'Orange' :

Orange is a sweet and delicious fruit that is very healthy.

Orange is also a good disease-fighter and we can eat it every day.

replace first 'Apple' to 'Orange' :

Orange is a sweet and delicious fruit that is very healthy.

Apple is also a good disease-fighter and we can eat it every day.

• تابع وابسته *split()*

این تابع وابسته برای جداسازی کلمات یک رشته از طریق مشخص کردن یک کاراکتر به عنوان کاراکتر جداساز استفاده می‌شود.

معمولًا زمانی که از این تابع وابسته برای جداسازی یک رشته استفاده می‌کنید، مقداری که برمی‌گرداند، یک فهرست از عناصری است که جدا شده است، می‌باشد.

این تابع وابسته در داخل پرانتز دو مقدار می‌گیرد که هر دو مقداری اختیاری می‌باشد. مقدار اول همان کاراکتر جداساز می‌باشد، و مقدار دوم، مقداری است که مشخص می‌کند که رشته به چند تعداد تقسیم و جدا شود و در فهرست قرار گیرد.

python example - strings.py

```
139 # String in Python (رشته‌ها در پایتون)
140 # Split()
141
142 # Example 1: split the fruits
143 fruits = "Apple,Orange,Banana,Pineapple,Cherry,Lemon"
144 fruits_list = fruits.split(',')
145 print(f"Before Splitting: {fruits}\nAfter Splitting: {fruits_list}", end='\n\n')
146
147 # Example 2: split the Cars
148 Cars = "Chevrolet*Dodge*Ford*BMW*Mercedes"
149 Cars_list = Cars.split('*')
150 print(f"Before Splitting: {Cars}\nAfter Splitting: {Cars_list}")
```

شكل ۱۳-۴ - مثالی از تابع وابسته *split*

خروجی شکل ۱۳-۴ :

Before Splitting: Apple,Orange,Banana,Pineapple,Cherry,Lemon

After Splitting: ['Apple', 'Orange', 'Banana', 'Pineapple', 'Cherry', 'Lemon']

*Before Splitting: Chevrolet*Dodge*Ford*BMW*Mercedes*

After Splitting: ['Chevrolet', 'Dodge', 'Ford', 'BMW', 'Mercedes']

۴-۲-۲-۴) توابع وابسته‌ای که برای جستجو در یک رشته، می‌توان برای رشته‌ها به کار برد

- `find()` تابع وابسته
- `index()` تابع وابسته

هر دو از این توابع وابسته برای جستجو در یک رشته به کار می‌روند و هر کدام سه مقدار در داخل پرانتز می‌گیرند که مقدار اول، ضروری است و مقادیر دوم و سوم اختیاری می‌باشد. مقدار اول، مقداری است که آن را در رشته جستجو می‌کند. مقدار دوم مشخص می‌کند که فرایند جستجو در رشته از چه اندیسی شروع شود. مقدار سوم نیز مشخص می‌کند فرایند جستجو در کدام اندیسی پایان یابد (مقداری که می‌گیرد جزء محدوده جستجو قرار نخواهد گرفت). به عبارت دیگر، با مشخص کردن مقادیر دوم و سوم، محدوده جستجو به صورت دلخواه تعیین می‌شود.

تفاوت توابع وابسته `find` و `index` در نتیجه و خروجی می‌باشد. اگر تابع وابسته `find` مقدار داده شده را در رشته پیدا نکند، نتیجه‌ای که بر می‌گرداند ۱ - می‌باشد، اما اگر تابع وابسته `index` مقدار داده شده در رشته را پیدا نکند، خطأ خواهد داد.

python example - strings.py

```
154 # String in Python (رشته‌ها در پایتون)  
155 # Find(value, start, end)  
156  
157 newString = "Apples are delicious fruits."  
158 # Words: A p p l e s   a r e   d e l i c i o u s   f r u i t s .  
159 # Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27  
160  
161 print(f"location of 'fruits' is: {newString.find("fruits")}")  
162 print(f"location of 'Apples' is: {newString.find("Apples")}")  
163 print(f"location of 'orange' is: {newString.find("orange")}")
```

شكل ۱۴-۴- مثالی از تابع وابسته *find*

خروجی شکل ۱۴-۴ :

location of 'fruits' is: 21

location of 'Apples' is: 0

location of 'orange' is: -1

```
168 # String in Python (رشته‌ها در پایتون)  
169 # Index(value, start, end)  
170  
171 newString = "Apples are delicious fruits."  
172  
173 # Words: A p p l e s   a r e   d e l i c i o u s   f r u i t s .  
174 # Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27  
175  
176 print(f"location of 'fruits' is: {newString.index("fruits")}")  
177 print(f"location of 'Apples' is: {newString.index("Apples")}")  
178 print(f"location of 'orange' is: {newString.index("orange")}")
```

شكل ۴-۱۵- مثالی از تابع وابسته *index*

خروجی شکل ۴-۱۵:

location of 'fruits' is: 21

location of 'Apples' is: 0

Traceback (most recent call last):

print(f"location of 'orange' is: {newString.index("orange")}")")

^^^^^^^^^^^^^^^^^

ValueError: substring not found

python example - strings.py

```
182 # String in Python (رشته‌ها در پایتون)  
183 # find() vs index()  
184  
185 new_string = "In the name of GOD. In GOD we trust."  
186  
187 # Words: In the name of G O D . ...  
188 # Index: 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 ...  
189  
190 print(f"location of 'GOD' is: {new_string.find("GOD", 0, 19)}")  
191 print(f"location of 'GOD' is: {new_string.find("Apple", 0, 19)}", end='\n\n')  
192 print(f"location of 'GOD' is: {new_string.index("GOD", 0, 19)}")  
193 print(f"location of 'GOD' is: {new_string.index("Apple", 0, 19)}")
```

شكل ۴-۱۶- مثالی از توابع وابسته‌ی *index* و *find* با اعمال مقادیر دوم و سوم هر تابع وابسته

خروجی شکل ۴-۱۶:

location of 'GOD' is: 15

location of 'GOD' is: -1

location of 'GOD' is: 15

Traceback (most recent call last):

print(f"location of 'GOD' is: {new_string.index("Apple", 0, 19)}")

^^^^^^^^^^^^^^^^^

ValueError: substring not found

۳-۲-۴) فهرست‌ها در پایتون

فهرست‌ها^{۳۸} در پایتون برای نگهداری انواع داده‌ها استفاده می‌شود به عبارت دیگر نوعی مخزن است که در آن انواع داده‌ها قرار می‌گیرد.

نکته: داخل یک فهرست، می‌تواند یک یا چندین اعداد، رشته‌ها، تاپل‌ها، مجموعه‌ها و حتی فهرست‌ها را قرار گیرد. هم‌چنان می‌توان دیکشنری‌ها، مجموعه‌ها، تاپل‌ها و آرایه‌ها را در داخل فهرست قرار داد.

برای ایجاد یک فهرست معمولاً از کروشه ([[]]) استفاده می‌شود، و داده‌ها در داخل کروشه قرار می‌گیرند. داده‌ها با کاما(,) از هم جدا می‌شوند.

۱-۳-۲-۴) تعریف و ایجاد فهرست‌ها در پایتون

متداول‌ترین روش ایجاد فهرست در پایتون، با استفاده از کروشه می‌باشد، اما می‌توان از توابع داخلی استفاده کرد.

- استفاده از کروشه([[]])

- استفاده از تابع داخلی list

ساختار کلی فهرست‌ها در پایتون:

Custom_List_Name = [item1, item2, ...]

در ادامه برای آشنایی با ساختار فهرست‌ها مثال‌هایی آورده شده است.

³⁸ Lists

python example - lists.py

```
1 # فهرست‌ها در پایتون (Lists in Python)
2 # Examples:
3
4 List_Example_1 = [20, 65, 2000, 1, 56, 9, 896, 0, -632, -96345, -1, -1000000]
5
6 List_Example_2 = ["Chevrolet", "Ford", "Dodge", "BMW", "Mercedes"]
7
8 List_Example_3 = [20, "Red", 1364, 98536, 1, "Yellow", "2416", "21g"]
9
10 List_Example_4 = ["Sky", "Green Earth", "This is a text.", "Is This Text?"]
11
12 List_Example_5 = [[1, 2], ['red', 'blue', 'green'], [100], ["Samsung"]]
13
14 List_Example_5 = [[200, 'red', 'blue'], ["Samsung", 23, 5, 634, 42]]
```

شكل ۴-۱۷- مثال‌هایی از ساختار فهرست‌ها در پایتون

داده‌هایی که در فهرست‌ها می‌توان قرار داد:

- رشته‌ها
- اعداد(اعداد صحیح، اعشاری و مختلط)
- فهرست‌ها (فهرست‌ها تو در تو)
- دیکشنری‌ها
- مجموعه‌ها
- تاپل‌ها
- (True/False) بولین‌ها

۴-۳-۲-۴) توابع وابسته‌ی مربوط به فهرست‌ها در پایتون

جدول ۴-۳ - جدول مربوط به توابع وابسته موجود در نوع داده‌ی فهرست

توضیحات	تابع وابسته
برای اضافه کردن فقط یک داده به آخر فهرست استفاده می‌شود. یک مقدار می‌گیرد.	append(value)
برای حذف کردن تمامی داده‌های فهرست استفاده می‌شود. مقداری نمی‌گیرد.	clear()
یک کپی از فهرست گرفته و مقدار کپی شده را برمی‌گرداند. مقداری نمی‌گیرد.	copy(list)
تعداد دفعات تکرار شده را با توجه به مقداری که گرفته، برمی‌گرداند. یک مقدار می‌گیرد.	count(value)
برای اضافه کردن <u>چندین داده</u> به آخر فهرست استفاده می‌شود. یک مقدار می‌گیرد.	
نکته: داده‌ها حتما باید در داخل یک فهرست یا توپل که قابلیت پیمایش دارد، قرار گیرد تا عملیات اضافه شدن صورت گیرد.	extend(list)
اندیس مقداری که گرفته، برمی‌گرداند. یک مقدار می‌گیرد.	index(value)
برای اضافه کردن یک داده به مکانی مشخص از فهرست. دو مقدار می‌گیرد. مقدار اول مربوط به مکانی است که می‌خواهید داده در آن مکان قرار گیرد و معمولاً با اندیس مشخص می‌شود. و مقدار دوم داده‌ای است که اضافه می‌شود.	insert(index, value)
برای حذف کردن یک داده مشخص از فهرست استفاده می‌شود.	pop(index)

یک مقدار می‌گیرد و آن هم اندیس مقداری که می‌خواهد حذف شود.

برای حذف کردن یک داده مشخص از فهرست استفاده می‌شود.

remove(value)

یک مقدار می‌گیرد و آن هم خود مقداری که می‌خواهد حذف شود.

ترتیب فهرست را معکوس می‌نماید.

reverse()

مقداری نمی‌گیرد.

برای مرتب سازی فهرست استفاده می‌شود.

دو مقدار اختیاری می‌گیرد.

مقدار اول True یا False است. اگر مقدار True بگیرد، فرایند مرتب‌سازی را به صورت

معکوس و از بزرگ به کوچک انجام می‌دهد و اگر False بگیرد که مقدار پیش‌فرض

sort(reverse, key)

می‌باشد، فرایند مرتب‌سازی را از کوچک به بزرگ انجام می‌دهد.

مقدار دوم، مقداری است که چگونگی مرتب‌سازی را از کاربر می‌گیرد. معمولاً این مقدار

توسط تابع lambda صورت می‌گیرد.

۴-۳-۲-۳) دسترسی به تعداد عناصر یک فهرست

برای اینکه تعداد عناصر یک فهرست را به دست آورید، باید از تابع len() استفاده کنید.

تابع () len از توابع داخلی پایتون هستند که در آینده به آن پرداخته خواهد شد.

python example - lists.py

```
18 # فهرست‌ها در پایتون (Lists in Python)
19 # Len(list)
20
21 List_Example_1 = [20, 65, 2000, 1, 56, 9, 896, 0, -632, -96345, -1, -1000000]
22 print(f"Length of List_Example_1 is {len(List_Example_1)}")
23
24 List_Example_2 = [20, "Red", 1364, 98536, 1, "Yellow", "2416", "21g"]
25 print(f"Length of List_Example_2 is {len(List_Example_2)}")
26
27 List_Example_3 = [[1, 2], ['red', 'blue', 'green'], [100], ["Samsung"]]
28 print(f"Length of List_Example_3 is {len(List_Example_3)}")
29
30 List_Example_4 = [[200, 'red', 'blue'], ["Samsung", 23, 5, 634, 42]]
31 print(f"Length of List_Example_4 is {len(List_Example_4)}")
```

شکل ۴-۱۸- به دست آوردن تعداد اعضای فهرست با استفاده از تابع *len*

خروجی شکل ۴-۱۸:

Length of List_Example_1 is 12

Length of List_Example_2 is 8

Length of List_Example_3 is 4

Length of List_Example_4 is 2

نکته: با توجه به دو مثال انتهایی شکل ۴-۱۸، اعضای درون فهرست، از فهرست‌ها مختلف تشکیل شده است،

بنابراین تعداد فهرست‌ها تشکیل شده مهم هستند و اعضای داخل فهرست‌ها در این قسمت مهم نیستند، چراکه

هر فهرستی که داخل فهرست اصلی قرار می‌گیرد به عنوان یک عضو از فهرست اصلی است و اعضای داخل فهرستها را محاسبه نمی‌شود.

برای محاسبه اعضای داخل فهرست‌ها، باید تک به تک فهرست‌های موجود در داخل فهرست اصلی را استخراج کرده و تعداد اعضای هر فهرست را توسط تابع `len` محاسبه کنید.

۴-۳-۲-۴) دسترسی به عناصر فهرست

برای این که یک عنصر خاصی از یک فهرست انتخاب کنید و در جاهای مختلف از کدتان استفاده کنید، می‌توانید از ساختار زیر استفاده کنید.

python example - lists.py

```
35 # Lists in Python ( فهرست‌ها در پایتون )
36 # دسترسی به یک عنصر خاص
37 # Variable_Name = List_Name[index]
38
39 List_Example_1 = [20, "Red", 1364, 98536, 1, "Yellow", "2416", "21g"]
40
41 select_item_1364 = List_Example_1[2]
42 print(select_item_1364)
43
44 select_item_yellow = List_Example_1[5]
45 print(select_item_yellow)
```

شکل ۴-۱۹- دسترسی به یک عنصر خاص فهرست

خروجی شکل ۱۹-۴:

1364

Yellow

همچنین می‌توانید یک محدوده مشخص از فهرست را انتخاب کرده و در یک فهرست دیگر قرار دهید و استفاده کنید.

python example - lists.py

```
49 # Lists in Python )  
50 دسترسی به یک محدوده خاص از یک لیست #  
51 # New_List_Name = List_Name[Start_Index:End_index:Step]  
52  
53 List_Example_1 = [20, "Red", 1364, 98536, 1, "Yellow", "2416", "21g"]  
54  
55 select_red_to_yellow = List_Example_1[1:6:2]  
56 print(f'select_red_to_yellow: {select_red_to_yellow}')  
57  
58 # Select 20, 1364, 1, "2416"  
59 select_custom = List_Example_1[::-2]  
60 print(f'select_custom: {select_custom}')
```

شکل ۴-۲۰- دسترسی به محدوده خاص از فهرست

خروجی شکل ۴-۲۰:

select_red_to_yellow: ['Red', 98536, 'Yellow']

select_custom: [20, 1364, 1, '2416']

نکته: در مثال آخر شکل ۵، تمامی اعضای فهرست انتخاب شدند (List_Example_1[:]) و سپس عدد ۲(عدد دلخواه) را به عنوان قدم قرار داده شده است، به این معناست که کل فهرست به صورت دو قدم دو قدم پیمایش می‌کند (List_Example_1[::-2]).

۴-۳-۲-۵) تغییر عناصر یک فهرست

برای تغییر یک عنصر خاص از یک فهرست می‌توانید با گرفتن اندیس آن عنصر و سپس استفاده از ساختار زیر، آن عنصر را حذف کرده و عنصر جدید را جایگزین کنید.

python example - lists.py

```
64 # Lists in Python ( فهرست‌ها در پایتون )
65 # Change List Item/Items
66 #     Change One Item of List
67 #     Syntax: List_Example[index] = new_Value
68
69 Car_List = ["Chevrolet", "Ford", "Dodge", "BMW", "Mercedes"]
70
71 # Before Changing
72 print(f"Before: {Car_List}")
73
74 Car_List[1] = "McLaren"
75
76 # After Changing
77 print(f"After: {Car_List}")
```

شکل ۴-۲۱-۴-مثالی از تغییر یکی از عناصر فهرست

خروجی شکل ۴-۲۱:

Before: ['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']

After: ['Chevrolet', 'McLaren', 'Dodge', 'BMW', 'Mercedes']

برای تغییر یک محدوده خاص از فهرست می‌توان از ساختار زیر استفاده کرد:

python example - lists.py

```
81 # Lists in Python ( فهرست‌ها در پایتون )
82 # Change List Item/Items
83 #     Change many Items of List
84 #     Syntax: List_Example[StartIndex:EndIndex:Step] = [new_Values]
85
86 Car_List = ["Chevrolet", "Ford", "Dodge", "BMW", "Mercedes"]
87 print(f"Before: {Car_List}")
88 Car_List[1:3] = ["McLaren", "Lamborghini"]
89 print(f"After: {Car_List}")
```

شکل ۴-۲۲- تغییر چند عنصر از فهرست

خروجی شکل ۴-۲۲:

Before: ['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']

After: ['Chevrolet', 'McLaren', 'Lamborghini', 'BMW', 'Mercedes']

یادآوری: مقدار EndIndex جزء محدوده انتخاب شده حساب نمی‌شود و محدوده از StartIndex شروع شده و

در EndIndex-1 پایان می‌یابد.

۶-۳-۲-۴) افزودن عناصر جدید به فهرست

برای این که یک عنصر یا مجموعه‌ای از عناصر را به فهرست اضافه کنید می‌توانید از روش‌های زیر استفاده کنید:

- استفاده از تابع وابسته `append` برای افزودن فقط یک عنصر به انتهای فهرست (شکل ۲۳-۴).
- استفاده از تابع وابسته `insert` برای افزودن یک عنصر به مکان دلخواه (شکل ۲۳-۴).
- استفاده از تابع وابسته `extend` برای افزودن مجموعه‌ای از عناصر به انتهای فهرست (شکل ۲۴-۴).

python example - lists.py

```
93 # Lists in Python ( )
94 # Adding new Item/Items to List
95 Car_List = ["Chevrolet", "Ford", "Dodge", "BMW", "Mercedes"]
96
97 # Add a new item to list by append() method
98 Car_List.append("McLaren")
99 print(f"Adding 'McLaren' in Car List by append():\n\t{Car_List}")
100
101 # Add a new item to list by insert() method
102 Car_List.insert(1, "Ferrari")
103 print(f"Adding 'Ferrari' in Car List by insert():\n\t{Car_List}")
104
```

شکل ۴-۲۳-۴) افزودن یک عنصر جدید به فهرست با استفاده از توابع وابسته `insert` و `append`

خروجی شکل ۴-۲۳:

Adding 'McLaren' in Car List by append():

`['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes', 'McLaren']`

Adding 'Ferrari' in Car List by insert():

```
['Chevrolet', 'Ferrari', 'Ford', 'Dodge', 'BMW', 'Mercedes', 'McLaren']
```

python example - lists.py

```
107 # Lists in Python ( فهرست‌ها در پایتون )
108 # Adding new Items to List
109
110 Car_List = ["Chevrolet", "Ford", "Dodge", "BMW", "Mercedes"]
111
112 # Add new items to list by extend() method
113 New_Car_List = ["McLaren", "Ferrari"]
114 Car_List.extend(New_Car_List)
115 #You can write like this:
116 #     Car_List.extend(["McLaren", "Ferrari"])
117
118 print(f"Adding ['McLaren', 'Ferrari'] in Car List by extend(): \n\t{Car_List}")
```

شكل ۲۴-۴-۴- افزودن عناصر جدید به فهرست با تابع وابسته *extend*

:۲۴-۴ خروجی

Adding ['McLaren', 'Ferrari'] in Car List by extend():

```
['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes', 'McLaren', 'Ferrari']
```

باتوجه به مثال شکل ۲۴-۴، طبق خط ۱۱۶ از کد، شما می‌توانید، به دو صورت از تابع وابسته *extend* استفاده

کنید:

✓ یک فهرست جداگانه ایجاد کرده و نام آن فهرست را در داخل تابع وابسته *extend* قرار دهید.

✓ داخل تابع وابسته `extend` فهرست ایجاد کرده و عناصری که می‌خواهید اضافه کنید، در داخل فهرست قرار دهید.

۷-۳-۲-۴) حذف عنصر/عناصر یک فهرست

از روش‌های زیر برای حذف عناصر فهرست استفاده می‌شود:

- استفاده از تابع وابسته `remove` برای حذف یک عنصر با استفاده از نام آن عنصر (شکل ۴-۲۵).
- استفاده از تابع وابسته `pop` برای حذف یک عنصر با استفاده از اندیس آن عنصر (شکل ۴-۲۵)..
- استفاده از کلیدواژه `del` برای حذف یک عنصر خاص (`del list_name[index]`) (شکل ۴-۲۵)..
- استفاده از تابع وابسته `clear` برای حذف تمامی عناصر یک فهرست (شکل ۴-۲۶).

python example - lists.py

```
122 # Lists in Python (فهرست‌ها در پایتون)
123 # Remove Item/Items from List
124
125 Car_List = ["Chevrolet", "Ford", "Dodge", "BMW", "Mercedes"]
126 print(f"Car List: \n\t{Car_List}")
127
128 # remove() method
129 Car_List.remove("Ford")
130 print(f"Remove 'Ford' from Car List by remove(): \n\t{Car_List}")
131
132 # pop() method
133 Car_List.pop(2) # remove 'BMW'
134 print(f"Remove 'BMW' from Car List by pop(): \n\t{Car_List}")
135
136 # del keyword
137 del Car_List[-1] # remove 'Mercedes'
138 print(f"Remove 'Mercedes' from Car List by del keyword: \n\t{Car_List}")
```

شکل ۴-۲۵- حذف عنصر با توابع وابسته `pop` و `remove` و کلیدواژه `del`

Car List:

['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']

Remove 'Ford' from Car List by remove():

['Chevrolet', 'Dodge', 'BMW', 'Mercedes']

Remove 'BMW' from Car List by pop():

['Chevrolet', 'Dodge', 'Mercedes']

Remove 'Mercedes' from Car List by del keyword:

['Chevrolet', 'Dodge']

python example - lists.py

```
142 # Lists in Python (فهرست‌ها در پایتون)
143 # Remove all items from List
144
145 Car_List = ['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']
146 print(f"Car List: \n\t{Car_List}")
147
148 # Remove all items by clear() method
149 Car_List.clear()
150 print(f"Remove all items from Car List by clear(): \n\t{Car_List}")
```

شکل ۴-۲۶- حذف تمامی عناصر با تابع وابسته *clear*

Car List:

`['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']`

Remove all items from Car List by clear():

`[]`

۴-۳-۲-۴) کپی‌گیری از فهرست

با توجه به شکل ۴-۲۷، دو روش برای کپی‌گیری از فهرست وجود دارد:

- استفاده از تابع وابسته `copy`

- استفاده از تابع داخلی `(new_List = list(old_List))`

python example - lists.py

```
154 # Lists in Python (فهرستها در پایتون)
155 # list Copy
156
157 Car_List = ['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']
158 print(f"Car List: \n\t{Car_List}")
159
160 # by copy() method
161 new_car_list_by_copy_method = Car_List.copy()
162 print(f"new_car_list_by_copy_method: \n\t{new_car_list_by_copy_method}")
163
164 # by build-in function
165 new_car_list_by_build_in_function = list(Car_List)
166 print(f"new_car_list_by_build_in_function: \n\t{new_car_list_by_build_in_function}")
```

شکل ۴-۲۷-۴) مثالی از کپی‌گیری از فهرست با روش تابع وابسته `copy` و تابع داخلی `list`

خروجی شکل ۴-۲۷:

Car List:

['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']

new_car_list_by_copy_method:

['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']

new_car_list_by_build_in_function:

['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']

۹-۳-۲-۴) فهرست‌های تو در تو

فهرست‌های تو در تو^{۳۹} یا همان فهرست‌های چندسطحی، یک ساختار داده پیشرفته در پایتون هستند که امکان ذخیره‌سازی فهرست‌ها به عنوان عناصر درون یک فهرست دیگر را فراهم می‌کنند. این ساختار برای نمایش داده‌های پیچیده مانند ماتریس‌ها، جداول، یا روابط سلسله‌مراتبی بسیار مفید و منظم است.

برای ایجاد فهرست‌های تو در تو، می‌توانید یک یا چند فهرست را به عنوان عناصر داخل یک فهرست قرار دهید.

برای مثال:

python example - lists.py

```
170 # # Lists in Python ( فهرست‌ها در پایتون )
171 # # Nested List
172
173 nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 20], ['red', 'blue', 'green'], [100], ["Samsung"]]
174 print(nested_list)
```

شکل ۴-۲۸-۴-مثالی از فهرست‌های تو در تو در پایتون

خروجی شکل ۴-۲۸-۴:

³⁹ Nested Lists

`[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 20], ['red', 'blue', 'green'], [100], ['Samsung']]`

۴-۳-۲-۱) دسترسی به عناصر فهرست‌های تو در تو

برای دسترسی به عناصر فهرست‌های تو در تو طبق شکل ۲۹-۴، از چندین اندیس استفاده می‌شود. هر اندیس نشان‌دهنده‌ی یک سطح از سلسله‌مراتب است.

python example - lists.py

```
178 # Lists in Python ( فهرست‌ها در پایتون )
179 # Access to nested list items
180
181 nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
182
183 print(nested_list[0])    # دسترسی به اولین زیرلیست
184
185 print(nested_list[1][2]) # دسترسی به عنصر سوم از زیرلیست دوم
```

شکل ۴-۲۹-۴-مثالی از دسترسی به عناصر فهرست‌های تو در تو در پایتون

خروجی شکل ۴-۲۹:

`[1, 2, 3]`

`6`

۴-۲-۳-۹-۲) افزودن و حذف عناصر در فهرست‌های تو در تو

طبق اشکال ۴ و ۳۰-۳۱، می‌توانید عناصر جدیدی به فهرست‌های تو در تو اضافه کنید یا آن‌ها را حذف نمایید.

python example - lists.py

```
189 # Lists in Python ) فهرست‌ها در پایتون( )
190 # Adding item/items to nested list
191
192 nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
193 print(f"nested_list: \n\t{nested_list}")
194
195 nested_list.append(1024) # اضافه کردن یک عنصر
196 print(f"Adding One item to nested_list: \n\t{nested_list}")
197
198 nested_list.append([10, 11, 12]) # اضافه کردن یک لیست
199 print(f"Adding One list to nested_list: \n\t{nested_list}")
200
201 nested_list.extend([2048, 4096, 8192]) # اضافه کردن چند عنصر
202 print(f"Adding many items to nested_list: \n\t{nested_list}")
203
204 nested_list.extend([('red', 'yellow', 'green'), ['blue', 'gray', 'green']]) # اضافه کردن چند لیست
205 print(f"Adding many lists to nested_list: \n\t{nested_list}")
```

شكل ۴-۳۰-۴-مثالی از افزودن یک یا چند عنصر به داخل فهرست‌های تو در تو

: ۴-۳۰ خروجی شکل

nested_list:

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

Adding One item to nested_list:

[[1, 2, 3], [4, 5, 6], [7, 8, 9], 1024]

Adding One list to nested_list:

[[1, 2, 3], [4, 5, 6], [7, 8, 9], 1024, [10, 11, 12]]

Adding many items to nested_list:

`[[1, 2, 3], [4, 5, 6], [7, 8, 9], 1024, [10, 11, 12], 2048, 4096, 8192]`

Adding many lists to nested_list:

`[[1, 2, 3], [4, 5, 6], [7, 8, 9], 1024, [10, 11, 12], 2048, 4096, 8192, ['red', 'yellow', 'green'], ['blue', 'gray', 'green']]`

python example - lists.py

```
209 # Lists in Python (فهرست‌ها در پایتون)  
210 # remove item/items to nested list  
211  
212 nested_list = [[1, 2, 3], [4, 5, 6], [7, 8, 9], 1024, [10, 11, 12], 2048, 4096, 8192]  
213 print(f"nested_list: \n\t{nested_list}")  
214  
215 nested_list.remove(1024)  
216 print(f"Remove one item from nested_list using remove() method: \n\t{nested_list}")  
217  
218 del nested_list[4]  
219 print(f"Remove one item from nested_list using index of item: \n\t{nested_list}")  
220  
221 del nested_list[1][2]  
222 print(f"Remove one item from sub lists of nested_list using index of item: \n\t{nested_list}")
```

شکل ۴-۳۱- مثالی از حذف عنصر از داخل فهرست‌های تو در تو

:۳۱-۴ خروجی

nested_list:

`[[1, 2, 3], [4, 5, 6], [7, 8, 9], 1024, [10, 11, 12], 2048, 4096, 8192]`

Remove one item from nested_list using remove() method:

`[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], 2048, 4096, 8192]`

Remove one item from nested_list using index of item:

`[[1, 2, 3], [4, 5, 6], [7, 8, 9], [10, 11, 12], 4096, 8192]`

Remove one item from sub lists of nested_list using index of item:

`[[1, 2, 3], [4, 5], [7, 8, 9], [10, 11, 12], 4096, 8192]`

۱۰-۳-۲-۴) ادغام کردن دو فهرست

باتوجه به شکل ۳۲-۴، برای ادغام کردن دو فهرست در یک فهرست دو روش وجود دارد:

- استفاده از حلقه‌ها.
- استفاده از تابع وابسته `.extend`

python example - lists.py

```
226 # Lists in Python (فهرست‌ها در پایتون)
227 # Merge Two List by for loop
228
229 first_list = ['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']
230 second_list = ["McLaren", "Ferrari"]
231 third_list = ["Nissan", "Porsche"]
232
233 print(f"first_list: {first_list}\nsecond_list: {second_list}")
234
235 # Using for loop to add second_list to first_list
236 for i in second_list:
237     first_list.append(i)
238 print(f"merge two list by for loop: \n\t{first_list}\n")
239
240 # using extend() method to add third_list to first_list
241 first_list.extend(third_list)
242 print(f"merge two list by extend(): \n\t{first_list}")
```

شکل ۴-۳۲-۴) ادغام کردن دو فهرست با حلقه `for` و تابع وابسته `extend`

first_list: ['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes']

second_list: ['McLaren', 'Ferrari']

merge two list by for loop:

['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes', 'McLaren', 'Ferrari']

merge two list by extend():

*['Chevrolet', 'Ford', 'Dodge', 'BMW', 'Mercedes', 'McLaren', 'Ferrari', 'Nissan',
'Porsche']*

۴-۲-۴) دیکشنری‌ها در پایتون

در پایتون، دیکشنری‌ها^{۴۰} یکی از انواع داده‌های بسیار مفید و پرکاربرد هستند. دیکشنری‌ها به شما اجازه می‌دهند تا داده‌ها را به صورت جفت‌های کلید-مقدار ذخیره کنید. برای درک ساده‌تر این موضوع به مثال زیر توجه نمایید:

فرض کنید فرمی وجود دارد که کاربر باید اطلاعاتی مانند نام و نامخانوادگی، کد ملی، سن، رنگ مورد علاقه و سایر جزئیات را تکمیل کند. هدف این است که این اطلاعات را ذخیره و مدیریت کنید. حال به بررسی روش‌های مختلف ذخیره‌سازی پرداخته می‌شود:

روش اول، استفاده از متغیرهای جداگانه: در این روش، برای هر داده‌ی کاربر، یک متغیر مستقل تعریف می‌شود.

- متغیری برای نام و نامخانوادگی

⁴⁰ Dictionaries(Dictionary)

• متغیری برای کد ملی

• متغیری برای سن و...

این روش برای حجم کم داده مناسب است، اما زمانی که تعداد فیلدهای فرم زیاد باشد (مثلاً ۲۰ فیلد)، مدیریت متغیرها بسیار دشوار می‌شود. علاوه بر این، احتمال اشتباه در استفاده یا فراخوانی متغیرها افزایش می‌یابد.

روش دوم، استفاده از فهرست‌ها: روش دیگر این است که داده‌های ورودی کاربر را به ترتیب در یک فهرست ذخیره کنید. برای مثال:

["علی", "آبی", ۱۲۳۴۵۶۷۸۹۰, ۲۵]

اگرچه این روش ساده‌تر است، اما مشکلی اساسی وجود دارد:

وابستگی به ترتیب داده‌ها.

برای مثال:

• اگر کاربر اطلاعاتی را وارد نکند (مانند رنگ مورد علاقه)، ترتیب داده‌ها به هم می‌ریزد.

• همچنین، برای دسترسی به هر داده باید به یاد داشته باشیم که آن داده در چه موقعیتی (اندیسی) در فهرست قرار دارد. این موضوع خوانایی و قابلیت نگهداری کد را کاهش می‌دهد.

روش سوم، استفاده از دیکشنری‌ها: دیکشنری یک ساختار داده مناسب برای ذخیره‌سازی اطلاعات مرتبط است که داده‌ها را به صورت کلید: مقدار ذخیره می‌کند. هر کلید نشان‌دهنده نوع داده (مانند "نام" یا "سن") و هر مقدار نمایانگر محتوای مرتبط با آن کلید است. برای مثال:

}

"نام": "علی",

"کد ملی": "۱۲۳۴۵۶۷۸۹۰"

"سن": ۲۵

"رنگ مورد علاقه": "آبی"

{

مزایای روش دیکشنری:

- داده‌ها مستقل از ترتیب ذخیره می‌شوند، بنابراین اگر کاربر اطلاعاتی را وارد نکند (مثلاً رنگ مورد علاقه)، سایر داده‌ها بدون مشکل ذخیره می‌شوند.
- دسترسی به داده‌ها بسیار ساده است؛ کافی است کلید مربوطه را بدانید. مثلاً برای دسترسی به "کد ملی"، کافی است از کلید "کد ملی" استفاده کنید.

۴-۲-۴) تعریف و ایجاد دیکشنری‌ها در پایتون

دیکشنری‌ها معمولاً با مقادیر جفت کلید^{۴۱} و مقدار^{۴۲} مشخص می‌شوند و اگر مقادیر زیادی جفت کلید و مقدار داشته باشید باید با کاما(،) از هم جدا کنید. نحوه ایجاد آن به دو صورت می‌باشد:

۱. با استفاده از آکولاد({})

۲. با استفاده از توابع داخلی پایتون.

ساختار کلی دیکشنری‌ها در پایتون

⁴¹ Key

⁴² Value

Custom_Dictionar_Name = {key1:value1, key2:value2}

در شکل ۴-۳۳، مثال‌هایی از دیکشنری‌ها در پایتون آورده شده است.

python example - dictionaris.py

```
1 # Dictionary in python (دیکشنری در پایتون)
2 """
3 Syntax:
4     my_dictionary = {
5         key:value
6     }
7 """
8 # Examples
9 dictionary_example_1 = {
10    "name": "Reza",
11    "age": 23,
12    "favorite color": "yellow"
13 }
14 dictionary_example_2 = {
15    "car_1": {
16        "Brand": "Chevrolet",
17        "Model": "Corvette"
18    },
19    "car_1": {
20        "Brand": "Dodge",
21        "Model": "Viper"
22    }
23 }
```

شکل ۴-۳۳- مثال‌هایی از دیکشنری‌ها در پایتون

نکات مهم در هنگام تعریف دیکشنری:

- ✓ معمولاً کلیدهای که در دیکشنری تعریف می‌کنید، می‌توانند به صورت رشته، عدد و تاپل‌ها تعریف شوند.

- ✓ مقادیری که به کلیدها می‌دهید می‌تواند از هر نوع داده باشد مانند رشته، عدد، فهرست، مجموعه‌ها، تاپل‌ها. حتی می‌توان یک دیکشنری دیگری را به عنوان مقدار یک کلید تعریف کرد.
- ✓ برای مقادیر کلیدها می‌توانید تابع نیز تعریف کنید همانند شکل ۳۴-۴ به عبارت دیگر یک تابع به عنوان مقدار برای کلید بدهید.
- ✓ معمولاً در تعریف کلید، باید آن را در داخل "" قرار دهید (مگر اینکه بصورت عددی باشد).
- ✓ کلیدهای تعریف شده باید منحصر به فرد باشند و کلیدهای مشابه وجود نداشته باشند.

python example - dictionaris.py

```

27 # Dictionary in python (دیکشنری در پایتون)
28 # Function => تابع
29 def Sum_Two_Number(a,b):
30     return a + b
31
32 dictionary_example_3 = {
33     1: "Hello",
34     2: "Good_bye",
35     "3": "This is String",
36     ("a", "b", "c"): 1,
37     ("d", "e", "f"): ["Chevrolet", "Dodge"],
38     "Car": {
39         "Brand": "Dodge",
40         "Model": "Viper"
41     },
42     "Sum_Function": Sum_Two_Number
43 }
44 x = dictionary_example_3["Sum_Function"](2,1)
45 print(x)

```

شكل ۴ - ۳۴ - تعریف تابع داخل دیکشنری و استفاده از تابع

۳

۲-۴-۲-۴) توابع وابسته مربوط به دیکشنری‌ها در پایتون

جدول ۴-۴- جدول مربوط به توابع وابسته موجود در نوع داده‌ی دیکشنری

توضیحات	تابع وابسته
پاک کردن تمامی عناصر(کلید و مقدار) دیکشنری مقداری نمی‌گیرد.	clear()
یک کپی از دیکشنری گرفته و مقدار کپی شده را برمی‌گرداند. مقداری نمی‌گیرد.	copy()
باقی‌گذاشتن مقداری که می‌گیرد یک دیکشنری با کلید و مقدار برمی‌گرداند.	
دو مقدار می‌گیرد. مقدار اول ضروری است و مربوط به کلیدهای دیکشنری است و باید در قالب Tuple یا List و یا ... که قابلیت پیمایش بر روی آیتم‌ها را دارد، مقداردهی کرد.	fromkeys(keys, values)
مقدار دوم اختیاری است و مربوط به مقادیر هر کلید می‌باشد و باید در قالب List یا Tuple و یا ... که قابلیت پیمایش بر روی آیتم‌ها را دارد، مقداردهی کرد.	
باقی‌گذاشتن مقداری که می‌گیرد، آن مقدار را در کلیدهای دیکشنری جستجو کرده و اگر کلیدی با همان مقداری که گرفته وجود داشته باشد، مقدار آن کلید را برمی‌گرداند.	get(keyName)
یک مقدار می‌گیرد.	

تمامی اجزای موجود در دیکشنری اعم از کلید و مقدار را در داخل فهرست به

صورت (key,value) برمی‌گرداند.

items()

مقداری نمی‌گیرد.

کلیدهای موجود در داخل دیکشنری را در داخل یک فهرست برمی‌گرداند.

keys()

مقداری نمی‌گیرد.

باتوجه به مقداری که می‌گیرد، آن مقدار را در کلیدهای دیکشنری جستجو کرده و

اگر کلیدی همانند مقداری که داده شده پیدا کند، کلید و مقدار آن را در

pop(keyName)

دیکشنری پاک می‌کند.

یک مقدار می‌گیرد.

آخرین کلید و مقدار در دیکشنری را پاک می‌کند.

popitem()

مقداری نمی‌گیرد.

باتوجه به مقداری که می‌گیرد، آن مقدار را در کلیدهای دیکشنری جستجو کرده،

اگر موجود باشد، مقدار آن کلید را در دیکشنری برمی‌گرداند. اگر کلید مشخص

شده وجود نداشته باشد یک مقدار از پیش تعریف شده را برمی‌گرداند.

دو مقدار می‌گیرد.

مقدار اول مقداری است در کلیدهای دیکشنری جستجو می‌کند.

setdefault(keyName, value)

مقدار دوم اختیاری است. اگر مقدار اول در کلیدهای دیکشنری موجود باشد و آن

کلید مقدار داشته باشد، این پارامتر کار خاصی انجام نمی‌دهد.

اگر مقدار اول در کلیدها موجود نباشد، مقدار تعریف شده در این پارامتر برگردانده

می‌شود.

مقدار پیش فرض این پارامتر None می‌باشد.

اگر بخواهید یک کلید و مقدارش را به دیکشنری اضافه کنید می‌توانید از این تابع

وابسته استفاده کنید.

update()

یک مقدار در داخل آکولاد به صورت ({"key": "value"}) می‌گیرد.

بر روی تمامی کلیدهای دیکشنری پیمایش کرده و مقدارشان را در داخل فهرست

قرار داده و آن فهرست را برمی‌گرداند.

values()

مقداری نمی‌گیرد.

۴-۲-۳) تعریف کلید و مقدار در دیکشنری

در قسمت قبل برای تعریف کردن کلید و مقدار جدید به دیکشنری از تابع وابسته update() استفاده شد. یک روش دیگر که متداول‌ترین روش برای تعریف کلید و مقدار است و حتی برای اینکه مقدار یک کلید را تغییر دهید به کار می‌رود، استفاده از ساختار زیر می‌باشد.

Dictionay_Custom_Name[key] = value

طبق ساختار بالا برای تعریف یک کلید و مقدار جدید، نام دیکشنری را قرار داده و سپس در داخل کروشه، نام کلیدی که می‌خواهید اضافه شود، نوشه می‌شود. سپس مقدار آن کلید را در قسمت value نوشه خواهد شد.

```
49 # Dictionary in python (دیکشنری در پایتون)
50 # define key:value for dictionary
51     # First method: insert to dictionary
52 Car_1 = {
53     "Brand": "Chevrolet",
54     "Model": "Corvette C8",
55     "Year": 2023,
56     "Colors": ["Red", "Green", "Blue", "Yellow"]
57 }
58 print(f"Information of Car_1 is:\n\t{Car_1}")
59
60 # define key:value for dictionary
61     # Second method: Car_Info[key] = value
62 Car_2 = {}
63 Car_2["Brand"] = "Ford"
64 Car_2["Model"] = "Mustang"
65 Car_2["Year"] = 2020
66 Car_2["Colors"] = ["Gray", "White", "Orange", "Black"]
67
68 print(f"Information of Car_2 is:\n\t{Car_2}")
```

شکل ۴-۳۵- مثالی از تعریف کلید و مقدار برای دیکشنری در پایتون

:۳۵-۴ خروجی شکل

Information of Car_1 is:

{'Brand': 'Chevrolet', 'Model': 'Corvette C8', 'Year': 2023, 'Colors': ['Red', 'Green', 'Blue', 'Yellow']}

Information of Car_2 is:

```
{'Brand': 'Ford', 'Model': 'Mustang', 'Year': 2020, 'Colors': ['Gray', 'White',  
'Orange', 'Black']}
```

۴-۲-۴) تغییر مقدار یک کلید در دیکشنری

برای این‌که مقدار یک کلید را تغییر دهید، می‌توانید از ساختار زیر استفاده کنید و در قسمت key مقدار کلیدی که می‌خواهید تغییر پیدا کند، نوشته می‌شود و سپس مقدار جدید را در قسمت value قرار خواهد گرفت.

(توجه کنید اگر مقدار کلید یافت نشود، یک کلید جدید و مقداری جدید به دیکشنری اضافه می‌شود. بنابراین نیاز است تا بزرگ و کوچک بودن حروف کلیدها مورد توجه قرار گیرد).

python example - dictionaris.py

```
72 # Dictionary in python (دیکشنری در پایتون)  
73 # Change value of a special key  
74 car_info = {  
75     'Brand': 'Chevrolet',  
76     'Model': 'Corvette C8',  
77     'Year': 2023  
78 }  
79 print(f"Unchanged Dictionary: \n\t{car_info}")  
80  
81 # If key is Exist:  
82 car_info["Year"] = 2024  
83 print(f"Change value of 'year' key: \n\t{car_info}")  
84  
85 # If key is NOT Exist:  
86 car_info["color"] = 'red'  
87 print(f"Define New Key and Value: \n\t{car_info}")
```

شکل ۴-۳۶- مثالی از تغییر مقدار یک کلید دیکشنری

Unchanged Dictionary:

{'Brand': 'Chevrolet', 'Model': 'Corvette C8', 'Year': 2023}

Change value of 'year' key:

{'Brand': 'Chevrolet', 'Model': 'Corvette C8', 'Year': 2024}

Define New Key and Value:

{'Brand': 'Chevrolet', 'Model': 'Corvette C8', 'Year': 2024, 'color': 'red'}

۴-۲-۴-۵) حذف کلید و مقدار در دیکشنری

برای حذف کلید و مقدارش چند روش وجود دارد.

روش اول که مربوط به تابع وابسته `pop` می‌باشد.

روش دوم مربوط به تابع وابسته `popitem` می‌باشد که آخرین کلید و مقدار را در دیکشنری حذف می‌کند.

روش سوم، استفاده از کلمه کلیدی `del` می‌باشد:

`del Dictionary_Custom_Name[key]`

روش چهارم، استفاده از تابع وابسته `clear` برای حذف تمامی کلیدها و مقادیر کلیدها در دیکشنری می‌باشد.

```
91 # Dictionary in python (دیکشنری در پایتون)
92 # Removing key:value from Dictionary
93 my_car_info = {
94     'Brand': 'Chevrolet',
95     'Model': 'Corvette C8',
96     'Year': 2024,
97     'color': 'red'
98 }
99 # pop() method
100 my_car_info.pop('color')
101 print(f"Remove key:value with pop method: \n\t{my_car_info}")
102
103 # popitem() method
104 my_car_info.popitem()
105 print(f"Remove key:value with popitem method: \n\t{my_car_info}")
106
107 # del keyword
108 del my_car_info["Model"]
109 print(f"Remove key:value with del keyword: \n\t{my_car_info}")
```

شكل ۴-۳۷- مثالی از حذف کلید و مقدار از دیکشنری با استفاده از توابع وابسته‌ی *pop* و *popitem* و کلیدواژه *del*

خروجی شکل ۴-۳۷:

Remove key:value with pop method:

{'Brand': 'Chevrolet', 'Model': 'Corvette C8', 'Year': 2024}

Remove key:value with popitem method:

{'Brand': 'Chevrolet', 'Model': 'Corvette C8'}

Remove key:value with del keyword:

{'Brand': 'Chevrolet'}

python example - dictionaris.py

```
113 # Dictionary in python (دیکشنری در پایتون)  
114 # Remove All key:value from Dictionary  
115  
116 my_car_info = {  
117     'Brand': 'Chevrolet',  
118     'Model': 'Corvette C8',  
119     'Year': 2024,  
120 }  
121 print(f"Before using clear method: \n\t{my_car_info}")  
122  
123 # clear method for remove every key and value of Dictionary  
124 my_car_info.clear()  
125  
126 print(f"After using clear method: \n\t{my_car_info}")  
127
```

شکل ۴-۳۸-مثالی از حذف تمامی کلید و مقدارها با استفاده از تابع وابسته *clear* در دیکشنری

خروجی شکل ۴-۳۸:

Before using clear method:

{'Brand': 'Chevrolet', 'Model': 'Corvette C8', 'Year': 2024}

After using clear method:

{}

۴-۲-۴) کپی گیری از دیکشنری

برای این که بتوانید یک کپی از دیکشنری بگیرید، دو روش وجود دارد:

روش اول، استفاده از تابع وابسته `copy` که در جدول ... توضیح داده شده است.

روش دوم، استفاده از توابع داخلی می‌باشد که در شکل ۳۹-۴ توضیح مختصری داده خواهد شد.

python example - dictionaris.py

```
130 # Dictionary in python (دیکشنری در پایتون)
131 # Copy of Dictionary
132
133 my_car_info = {
134     'Brand': 'Chevrolet',
135     'Model': 'Corvette C8',
136     'Year': 2024,
137 }
138 print(f"Main Dictionary: \n\t{my_car_info}")
139
140 # Using copy method
141 car_info_copy_1 = my_car_info.copy()
142 print(f"Copy of my_car_info by copy method: \n\t{car_info_copy_1}")
143
144 # Using dict build-in function
145 car_info_copy_2 = dict(my_car_info)
146 print(f"Copy of my_car_info by dict build-in: \n\t{car_info_copy_2}")
```

شکل ۴-۳۹-مثالی از کپی گیری از دیکشنری با تابع وابسته `copy` و تابع داخلی `dict`

خروجی: ۳۹-۴

Main Dictionary:

{'Brand': 'Chevrolet', 'Model': 'Corvette C8', 'Year': 2024}

Copy of my_car_info by copy method:

```
{'Brand': 'Chevrolet', 'Model': 'Corvette C8', 'Year': 2024}
```

Copy of my_car_info by dict build-in:

```
{'Brand': 'Chevrolet', 'Model': 'Corvette C8', 'Year': 2024}
```

۴-۲-۴) دیکشنری‌های تو در تو

دیکشنری‌های تو در تو^{۴۳} ساختار داده‌ای در پایتون هستند که به شما امکان می‌دهند دیکشنری‌ها را به عنوان مقدار درون دیکشنری‌های دیگر قرار دهید. این ساختار برای نمایش داده‌های پیچیده، مانند پایگاه داده‌های کوچک یا مدل‌سازی روابط چندسطحی، بسیار مفید است.

برای ایجاد یک دیکشنری تو در تو، می‌توانید دیکشنری‌های دیگر را به عنوان مقدار برای کلیدها در یک دیکشنری اصلی تعریف کنید.

python example - dictionaris.py

```
150 # # Dictionary in python (دیکشنری در پایتون)
151 # # Nested Dictionary Example
152 nested_dict = {
153     "person1": {"name": "Reza", "age": 23},
154     "person2": {"name": "Ali", "age": 22},
155     "person3": {"name": "Mohammad", "age": 24}
156 }
157 print(nested_dict)
```

شکل ۴-۴۰- مثالی از دیکشنری‌های تو در تو در پایتون

⁴³ Nested Dictionaries

```
{  
  'person1': {'name': 'Reza', 'age': 23},  
  'person2': {'name': 'Ali', 'age': 22},  
  'person3': {'name': 'Mohammad', 'age': 24}  
}
```

۴-۲-۴-۷-۱) دسترسی به مقادیر در دیکشنری‌های تو در تو

برای دسترسی به مقادیر در دیکشنری‌های تو در تو، می‌توانید از چندین کلید به ترتیب استفاده کنید. به شکل ۴-۱ توجه کنید.

```
161 # Dictionary in python (دیکشنری در پایتون)
162 # Access to Nested Dictionary items
163 nested_dict = {
164     "person1": {"name": "Reza", "age": 23},
165     "person2": {"name": "Ali", "age": 22},
166     "person3": {"name": "Mohammad", "age": 24}
167 }
168
169 # دسترسی به دیکشنری مربوط به شخص اول
170 print(f"Person1 info is: \n\t{nested_dict['person1']}")  

171
172 # دسترسی به نام شخص دوم
173 print(f"Name of person2 is: \n\t{nested_dict['person2']['name']}")  

174
175 # دسترسی به سن شخص سوم
176 print(f"Age of person3 is: \n\t{nested_dict['person3']['age']}")
```

شکل ۴-۴۱- مثالی از دسترسی به مقادیر در دیکشنری‌های تو در تو

خروجی شکل ۴۱-۴:

Person1 info is:

{'name': 'Reza', 'age': 23}

Name of person2 is:

Ali

Age of person3 is:

24

۴-۲-۴-۷-۲) افزودن و به روزرسانی در دیکشنری های تو در تو

شما می توانید به راحتی کلیدها و مقادیر جدیدی به دیکشنری های تو در تو اضافه کنید یا مقادیر موجود را طبق شکل ۴-۴ تغییر دهید.

python example - dictionaris.py

```
180 # Dictionary in python (دیکشنری در پایتون)
181 # Add item/items to nested dictionary
182
183 nested_dict = {
184     "person1": {"name": "Reza", "age": 23},
185     "person2": {"name": "Ali", "age": 22},
186     "person3": {"name": "Mohammad", "age": 24}
187 }
188 print(f"Main Dictionary: \n\t{nested_dict}")
189
190 افزودن شهر به اطلاعات شخص اول # افزودن شهر به اطلاعات شخص اول
191 nested_dict["person1"]["city"] = "Tabriz"
192 print(f"Update 1: \n\t{nested_dict}")
193
194 افزودن شخص جدید # افزودن شخص جدید
195 nested_dict["person4"] = {"name": "Diana", "age": 28}
196 print(f"Update 2: \n\t{nested_dict}")
```

شکل ۴-۴-۴-۲-۴-۲-۴-۷-۲) تعریف کلید و مدار در دیکشنری های تو در تو

خروجی شکل ۴-۴-۴-۲-۴-۲-۴-۷-۲)

Main Dictionary:

{'person1': {'name': 'Reza', 'age': 23}, 'person2': {'name': 'Ali', 'age': 22},

'person3': {'name': 'Mohammad', 'age': 24}}

Update 1:

```
{'person1': {'name': 'Reza', 'age': 23, 'city': 'Tabriz'}, 'person2': {'name': 'Ali',  
'age': 22}, 'person3': {'name': 'Mohammad', 'age': 24}}
```

Update 2:

```
{'person1': {'name': 'Reza', 'age': 23, 'city': 'Tabriz'}, 'person2': {'name': 'Ali',  
'age': 22}, 'person3': {'name': 'Mohammad', 'age': 24}, 'person4': {'name': 'Diana',  
'age': 28}}
```

۴-۲-۴-۷-۳) حذف از دیکشنری‌های تو در تو

برای حذف مقادیر از دیکشنری‌های تو در تو، می‌توانید از دستور `del` استفاده کنید.

مثال:

```
200 # Dictionary in python (دیکشنری در پایتون)  
201 # remove item/items from nested dictionary  
202  
203 nested_dict = {  
204     "person1": {"name": "Reza", "age": 23, "city": "Tabriz"},  
205     "person2": {"name": "Ali", "age": 22},  
206     "person3": {"name": "Mohammad", "age": 24},  
207     "person4": {"name": "Diana", "age": 28}  
208 }  
209 print(f"Main Dictionary: \n\t{nested_dict}")  
210  
211 # حذف شهر از اطلاعات شخص اول  
212 del nested_dict["person1"]["city"]  
213 print(f"Update 1: \n\t{nested_dict}")  
214  
215  
216 # حذف اطلاعات شخص چهارم  
217 del nested_dict["person4"]  
218 print(f"Update 2: \n\t{nested_dict}")
```

شکل ۴-۴- حذف کلید و مقدار در دیکشنری‌های تو در تو

: ۴۳-۴ خروجی شکل

Main Dictionary:

```
{'person1': {'name': 'Reza', 'age': 23, 'city': 'Tabriz'}, 'person2': {'name': 'Ali',  
'age': 22}, 'person3': {'name': 'Mohammad', 'age': 24}, 'person4': {'name': 'Diana',  
'age': 28}}
```

Update 1:

```
{'person1': {'name': 'Reza', 'age': 23}, 'person2': {'name': 'Ali', 'age': 22},  
'person3': {'name': 'Mohammad', 'age': 24}, 'person4': {'name': 'Diana', 'age': 28}}
```

Update 2:

```
{'person1': {'name': 'Reza', 'age': 23}, 'person2': {'name': 'Ali', 'age': 22},  
'person3': {'name': 'Mohammad', 'age': 24}}
```

۴-۲-۵) تاپل‌ها در پایتون

تاپل‌ها^{۴۴} برای ذخیره چندین عنصر(داده) در یک متغیر استفاده می‌شوند. نوع داده تاپل‌ها در پایتون شبیه فهرست‌ها می‌باشد با این تفاوت که تاپل‌ها تغییرناپذیرند^{۴۵} و نمی‌توان اعضای آن را تغییر دهید. از آنجا که تاپل‌ها تغییرناپذیر می‌باشند، دستورات حلقه با تاپل‌ها سریع‌تر از فهرست‌ها انجام می‌شود(در جلسات آینده توضیح داده خواهد شد).

اگر کاربر داده‌هایی داشته باشند که نباید تغییری کند، پیاده‌سازی آن‌ها با استفاده از تاپل‌ها، تضمین می‌کند که داده‌ها در مقابل تغییر امن هستند و امکان تغییر داده‌ها وجود ندارد و فقط قابلیت خواندن دارند.

می‌توان از قابلیت تغییرناپذیری عناصر در تاپل‌ها استفاده کرد و به عنوان کلیدهای دیکشنری استفاده کرد. تفاوت دیگری که تاپل‌ها با فهرست‌ها دارد در این است که برای ایجاد فهرست‌ها از کروشه یا براکت ([]) استفاده می‌شود اما برای تعریف و ایجاد تاپل‌ها از پرانتز () استفاده می‌شود. تاپل‌ها همانند فهرست‌ها می‌توانند عناصر تکراری نیز داشته باشند.

⁴⁴ Tuples

⁴⁵ Immutable

به طور خلاصه ویژگی‌هایی که تاپل‌ها دارد عبارت است از:

- مرتب شده^{۴۶}: موارد دارای ترتیب مشخصی هستند و این ترتیب تغییر نمی‌کند.
- تغییرناپذیر^{۴۷}: پس از ایجاد تاپل، امکان ایجاد تغییر، حذف و اضافه در تاپل‌ها وجود ندارد.
- قابلیت تکراری بودن^{۴۸}: عناصری که در تاپل‌ها وجود دارد، می‌توانند تکراری نیز باشند.

۴-۵-۲-۱) تعریف و ایجاد تاپل‌ها در پایتون

برای تعریف تاپل‌ها در پایتون از دو روش زیر می‌توان استفاده کرد:

- استفاده از پرانتز جهت ایجاد تاپل
- استفاده از تابع داخلی tuple برای ایجاد تاپل (عناصر باید داخل یک فهرست یا تاپل یا داده‌ای که قابلیت پیمایش دارد، قرار گیرد).

⁴⁶ Ordered

⁴⁷ Unchangeable

⁴⁸ Allow Duplicates

```
1 # تاپل در پایتون (Tuples in Python)
2 # روش‌های ایجاد تاپل
3
4 # first method: using ()
5 create_tuple_1 = ("Reza", "Mahdi", "Mohammad", "Ali")
6 print(f"Create tuple using (): \n\t{create_tuple_1}")
7 print(f"Type of create_tuple_1 is: \n\t{type(create_tuple_1)}")
8
9
10 # Second Method: use tuple build-in function
11 create_tuple_2 = tuple(("Reza", "Ali", "Mohammad", "Mahdi"))
12 print(f"Create tuple using tuple() build-in function : \n\t{create_tuple_2}")
13 print(f"Type of create_tuple_2 is: \n\t{type(create_tuple_2)}")
```

شکل ۴-۴-۴- تعریف و ایجاد تاپل‌ها در پایتون

خروجی شکل ۴-۴:

Create tuple using ():

('Reza', 'Mahdi', 'Mohammad', 'Ali')

Type of create_tuple_1 is:

<class 'tuple'>

Create tuple using tuple() build-in function :

('Reza', 'Ali', 'Mohammad', 'Mahdi')

Type of create_tuple_2 is:

<class 'tuple'>

نوع داده‌هایی که می‌توان در تاپل قرار داد:

➤ داده‌های عددی

➤ داده‌های رشته‌ای

➤ فهرست‌ها

➤ دیکشنری‌ها

➤ مجموعه‌ها

➤ تاپل‌ها (تاپل‌های تو در تو همانند فهرست‌های تو در تو)

➤ (True/False) بولین‌ها

```

17 # تاپل در پایتون (Tuples in Python)
18 # داده‌هایی که می‌توان در تاپل قرار داد
19
20 tuple_example_1 = (
21     "Reza",
22     23, 188.65, 10+5j,
23     ["Yellow", "Green", "Red", "Orange", [1, 2, 3, 4, 5]],
24     ('chevrolet', 'Dodge', 'Ford'),
25     True, False,
26     {"Abolfazl", "Amir", "Ehsan"},
27     {
28         "book": "Python",
29         "level": "Intermediate",
30         "rating": [5, 4.5, 4, 5, 4]
31     }
32 )
33 print(tuple_example_1)

```

شکل ۴-۴۵-مثالی از قرار دادن انواع داده‌ها در تاپل‌ها

خروجی شکل ۴-۴۵:

('Reza', 23, 188.65, (10+5j), ['Yellow', 'Green', 'Red', 'Orange', [1, 2, 3, 4, 5]], ('chevrolet', 'Dodge', 'Ford'), True, False, {'Amir', 'Abolfazl', 'Ehsan'}, {'book': 'Python', 'level': 'Intermediate', 'rating': [5, 4.5, 4, 5, 4]})

۴-۲-۵-۲-۴) تعداد عناصر موجود در تاپل‌ها

برای به دست آوردن تعداد عناصر یک تاپل می‌توان از تابع `len()` همانند فهرست‌ها، استفاده کرد.

```
37 # Tuples in Python (تاپل در پایتون)
38 # Length of Tuple
39
40 tuple_example_1 = (
41     "Yellow",
42     "Green",
43     "Red",
44     "Orange",
45     True,
46     1253,
47     52.364958,
48     ('chevrolet', 'Dodge', 'Ford')
49 )
50 print(f"Elements of tuple_example_1 is: \n\t{tuple_example_1}")
51 print(f"Length of tuple_example_1 is: {len(tuple_example_1)}")
```

شکل ۴-۴- به دست آوردن طول یک تاپل با استفاده از تابع *len*

خروجی شکل ۴-۴:

Elements of tuple_example_1 is:

('Yellow', 'Green', 'Red', 'Orange', True, 1253, 52.364958, ('chevrolet', 'Dodge', 'Ford'))

Length of tuple_example_1 is: 8

نکته: توجه داشته باشد که اگر بخواهید یک تاپل با یک عنصر ایجاد کنید، حتما اطمینان حاصل کنید که بعد از عنصر موردنظر در داخل تاپل، کاما گذاشته شده باشد، در غیر این صورت نوع داده‌ای که تعریف کرده‌اید تاپل نیست. برای درک بهتر این موضوع به شکل ۴۷-۴ توجه کنید.

python example - tuples.py

```
55 # تاپل در پایتون)
56 # Tuple With ONE Element
57
58 # This is Tuple
59 tuple_example_2 = ("Yellow")
60 print(f"tuple_example_2: {tuple_example_2}")
61 print(f"Type of tuple_example_2 is {type(tuple_example_2)}") # tuple
62
63 # This is NOT Tuple
64 tuple_example_3 = ("Yellow")
65 print(f"tuple_example_3: {tuple_example_3}")
66 print(f"Type of tuple_example_3 is {type(tuple_example_3)}") # string
```

شکل ۴-۴۷-۴- مثالی از تاپل تک عنصری

خروجی شکل ۴-۴۷-۴:

tuple_example_2: ('Yellow',)

Type of tuple_example_2 is <class 'tuple'>

tuple_example_3: Yellow

Type of tuple_example_3 is <class 'str'>

۴-۵-۲-۴) توابع وابسته‌ی مربوط به تاپل‌ها در پایتون

طبق جدول ۴-۵، توابع وابسته‌ی موجود در تاپل‌ها به دلیل غیرقابل تغییرپذیری بسیار اندک می‌باشد.

جدول ۴-۵- جدول مربوط به توابع وابسته موجود در نوع داده‌ی تاپل

تابع وابسته	توضیحات	مثال
count(value)	بازدید از مقداری که می‌گیرد، تعداد دفعاتی که آن مقدار در تاپل تکرار شده است را بر می‌گرداند.	Tuple_example = ("BMW", "Benz", "Ford", "Benz", "Ford", "Benz") X = Tuple_example.count("Benz") print(X) # 3
Index(value)	بازدید از عنصر مشخص شده پیدا کند، اندیس آن مقدار را از تاپل بر می‌گرداند.	Tuple_example = ("BMW", "Benz", "Ford", "Benz", "Ford", "Benz") X = Tuple_example.index("Ford") print(X) # 2

۴-۵-۲-۴) دسترسی به عناصر تاپل‌ها

برای دسترسی به یک عنصر مشخص از تاپل، کافی است اندیس آن عنصر را پیدا کرده و همانند شکل ۴-۴۸، به آن عنصر دسترسی داشته باشید.

```
70 # Tuples in Python (تاپل در پایتون)
71 # Access The Tuple Elements => variable = tuple_name[index]
72
73 tuple_example_4 = (
74     "Reza",
75     23,
76     188.65,
77     ["Yellow", "Green", "Red", "Orange"],
78     ('chevrolet', 'Dodge', 'Ford'),
79     True
80 )
81 print(f"Main Tuple: \n\t{tuple_example_4}")
82
83 # Extract list from tuple_example_4
84 print(f"List extract from tuple_example_4: \n\t{tuple_example_4[3]}")
85
86 # Extract float number from tuple_example_4
87 print(f"Float number extract from tuple_example_4: \n\t{tuple_example_4[2]}")
```

شکل ۴-۴-۴۸- مثالی از دسترسی به عناصر یک تاپل

خروجی شکل ۴-۴:

Main Tuple:

('Reza', 23, 188.65, ['Yellow', 'Green', 'Red', 'Orange'], ('chevrolet', 'Dodge', 'Ford'), True)

List extract from tuple_example_4:

['Yellow', 'Green', 'Red', 'Orange']

Float number extract from tuple_example_4:

188.65

- اندیس منفی: اگر مقدار اندیس را منفی دهید، شمارش را از آخر تاپل شروع می‌کند. برای مثال وقتی اندیس ۱ را استفاده می‌کنید، آخرین عنصر از تاپل را برمی‌گرداند، وقتی اندیس ۲- را استفاده می‌کنید، دومین عنصر از آخرین عنصر از تاپل را برمی‌گرداند. برای درک بیشتر این موضوع به شکل ۴-۴ توجه کنید:

python example - tuples.py

```
91 # Tuples in Python)
92 # Negative Index in Tuples
93 tuple_example_5 = (
94     "Yellow",
95     ("Green",),
96     True,
97     1253,
98     52.364958,
99     ('chevrolet', 'Dodge', 'Ford')
100 )
101 print(f"Main Tuple: \n\t{tuple_example_5}")
102
103 # Extract last tuple from tuple_example_5
104 last_item_of_tuple_example_5 = tuple_example_5[-1]
105 print(f"Last item of tuple_example_5 (tuple_example_5[-1]) is: \n\t{last_item_of_tuple_example_5}")
106
107 # Extract Int Number from tuple_example_5
108 myIntNumber = tuple_example_5[-3]
109 print(f"Int Number of tuple_example_5 (tuple_example_5[-3]) is: \n\t{myIntNumber}")
```

شکل ۴-۴- مثالی از اندیس منفی در تاپل‌ها

: ۴-۴ خروجی شکل

Main Tuple:

('Yellow', ('Green',), True, 1253, 52.364958, ('chevrolet', 'Dodge', 'Ford'))

Last item of tuple_example_5 (tuple_example_5[-1]) is:

('chevrolet', 'Dodge', 'Ford')

Int Number of tuple_example_5 (tuple_example_5[-3]) is:

نکته: برای دسترسی به یک محدوده مشخص از عناصر یک تاپل، با مشخص کردن اندیس اولیه و اندیس ثانویه، به آن محدوده دسترسی پیدا کنید.

یادآوری: توجه داشته باشید، مقداری که به اندیس ثانویه می‌دهید، جزء محدوده قرار نمی‌گیرد.

نکته: می‌توان از اندیس منفی برای دسترسی به محدوده خاص استفاده کرد.

python example - tuples.py

```

113 # تاپل در پایتون)
114 # Access Range of Tuple Elements
115 tuple_example_6 = (
116     "Red",
117     "Green",
118     True,
119     5639,
120     52.3649,
121     False
122 )
123 print(f"Main Tuple: \n\t{tuple_example_6}")
124
125 # Extract from "Green" to 52.3649
126 extract_range_of_tuple = tuple_example_6[1:5]
127 print(f"Extract data is: \n\t{extract_range_of_tuple}")
128 print(f"Type of extract data is: {type(extract_range_of_tuple)}")

```

شكل ۴-۵۰- مثال‌هایی از دسترسی به محدوده مشخص از تاپل

خروجی شکل ۴-۵۰:

Main Tuple:

('Red', 'Green', True, 5639, 52.3649, False)

Extract data is:

('Green', True, 5639, 52.3649)

Type of extract data is: <class 'tuple'>

۴-۵-۲-۴) بررسی وجود داشتن یک عنصر داخل تاپل

برای این که موجود بودن یک عنصر را در داخل یک تاپل بررسی کنید، نیاز به کلیدواژه `in` دارید. مقداری که برمی‌گرداند، یک بولین می‌باشد و مقادیر `True` یا `False` را برمی‌گرداند.

این ویژگی را می‌توانید در فهرست‌ها و مواردی که قابلیت پیمایش دارند، اعمال کنید.

python example - tuples.py

```
132 # Tuples in Python (تاپل در پایتون)
133 # Is Exist Or NOT ??? (بررسی وجود داشتن یک عنصر داخل تاپل)
134
135 tuple_example_7 = tuple(("Reza", "Ali", "Mohammad", "Mahdi", "Amir"))
136
137 print(f"tuple_example_7: \n\t{tuple_example_7}")
138
139 print(f"Is 'Reza' on the tuple_example_7: {"Reza" in tuple_example_7}") # True
140
141 print(f"Is 'Zahra' on the tuple_example_7: {"Zahra" in tuple_example_7}") # False
142
143 print(f"Is 'ali' on the tuple_example_7: {"ali" in tuple_example_7}") # False
144
145 print(f"Is 'Ehsan' on the tuple_example_7: {"Ehsan" in tuple_example_7}") # False
146
147 print(f"Is 'Mohammad' on the tuple_example_7: {"Mohammad" in tuple_example_7}") # True
```

شکل ۴-۵۱-۴) مثالی‌هایی از بررسی موجود بودن یک عنصر در داخل یک تاپل

خروجی شکل ۴-۵۱:

tuple_example_7:

('Reza', 'Ali', 'Mohammad', 'Mahdi', 'Amir')

Is 'Reza' on the tuple_example_7: True

Is 'Zahra' on the tuple_example_7: False

Is 'ali' on the tuple_example_7: False

Is 'Ehsan' on the tuple_example_7: False

Is 'Mohammad' on the tuple_example_7: True

۴-۵-۶) تغییر عناصر یک تاپل

تاپل‌ها تغییر ناپذیرند و به همین دلیل هیچ تابع وابسته برای تغییر عناصر وجود ندارد. بنابراین برای این که عناصر یک تاپل را تغییر دهید، دو راه وجود دارد:

۱. روش تبدیل کردن تاپل به فهرست: ابتدا باید آن تاپل را به فهرست تبدیل کرده و تغییر خود را اعم از افزودن، حذف و ... اعمال کرده و درنهایت دوباره تبدیل به تاپل کنید.

✓ برای تبدیل به فهرست از تابع داخلی `list` استفاده کنید.

✓ برای تبدیل به تاپل از تابع داخلی `tuple` استفاده کنید.

```
151 # تاپل در پایتون (Tuples in Python)
152 # Change Element of Tuples (تغییر عناصر تاپل)
153
154 # First Method: Convert Tuple to List
155 tuple_example_8 = ("Yellow", "Green", "Red", "Orange")
156 print(f"1. tuple_example_8: \n\t{tuple_example_8}")
157
158 # Convert tuple to list
159 tuple_to_list = list(tuple_example_8)
160 print(f"2. Convert Tuple To List: \n\t{tuple_to_list}")
161
162 # Now You Can Change
163 tuple_to_list[1] = "Blue"
164 print(f"3. Change Time (Blue instead of Green): \n\t{tuple_to_list}")
165
166 # After Change, Convert list to tuple
167 list_to_tuple = tuple(tuple_to_list)
168 print(f"4. Convert list to tuple: \n\t{list_to_tuple}")
```

شکل ۴-۵۲- مثالی از تغییر عناصر تاپل با روش تبدیل تاپل به فهرست

خروجی شکل ۴-۵۲:

1. *tuple_example_8:*

('Yellow', 'Green', 'Red', 'Orange')

2. *Convert Tuple To List:*

['Yellow', 'Green', 'Red', 'Orange']

3. *Change Time (Blue instead of Green):*

['Yellow', 'Blue', 'Red', 'Orange']

4. *Convert list to tuple:*

('Yellow', 'Blue', 'Red', 'Orange')

۲. افزودن تاپل به تاپل دیگر: برای اضافه کردن یک تاپل به تاپل دیگر می‌توانید از روش زیر استفاده کنید.

python example - tuples.py

```
172 # Tuples in Python (تاپل در پایتون)
173 # Change Element of Tuples (تغییر عناصر تاپل)
174
175 # Second Method: use addition assignment(+=)
176 tuple_example_9 = ("Yellow", "Green", "Red", "Orange")
177 print(f'Before Adding: {tuple_example_9}')
178
179 another_tuple = ("Blue",)
180 tuple_example_9 += another_tuple
181 print(f'After Adding: {tuple_example_9}')
```

شکل ۴-۵۳-مثالی از اضافه کردن یک تاپل به تاپل دیگر

خروجی شکل ۴-۵۳:

Before Adding: ('Yellow', 'Green', 'Red', 'Orange')

After Adding: ('Yellow', 'Green', 'Red', 'Orange', 'Blue')

۶-۲-۴) مجموعه‌ها در پایتون

مجموعه‌ها^{۴۹} نوع دیگری از انواع داده‌ها در پایتون می‌باشد که شبیه به مجموعه‌ها در دروس ریاضی می‌باشد. مجموعه‌ها نیز همانند فهرست‌ها و تاپل‌ها برای ذخیره چندین مقدار درون یک متغیر استفاده می‌شوند. در ادامه به ویژگی‌های مجموعه‌ها در پایتون اشاره می‌شود.

- مجاز نبودن عناصر یکسان: نمی‌توان عناصر یکسان را در درون مجموعه‌ها قرار داد و در صورت قرار دادن چندین عنصر یکسان در درون مجموعه‌ها، فقط یک از آن عناصر باقی خواهد ماند و بقیه عناصر مشابه حذف خواهند شد.
- نداشتن ترتیب عناصر: مجموعه‌ها ترتیب خاصی برای عناصر خود ندارند و نمی‌توان به صورت مستقیم به عناصر با استفاده از اندیس آن عنصر دسترسی پیدا کرد.
- عدم تغییر عناصر بعد از تعریف: بعد از تعریف عناصر مجموعه‌ها، امكان تغییرپذیری عناصر وجود ندارد. توجه داشته باشید که می‌توانید، عناصری به مجموعه اضافه و یا حذف کنید اما امكان تغییر عناصر وجود ندارد.

نکته:

- ✓ اگر داده‌های درون مجموعه‌ها فقط از نوع عددی باشند و هم‌چنین مثبت (اعشاری نیز می‌توانند باشند) نیز باشند، با خروجی گرفتن از مجموعه، اعداد به صورت مرتب خروجی گرفته می‌شوند.
- ✓ درمورد مقادیر رشته‌ای نیز بصورت الفبای انگلیسی مرتب خواهد شد.

۴-۲-۶) تعریف و ایجاد مجموعه‌ها در پایتون

برای تعریف مجموعه‌ها در پایتون طبق شکل ۴-۵ از دو روش می‌توان استفاده کرد:

- روش اول، استفاده از علامت آکولاد (مشابه دیکشنری‌ها)

- روش دوم، استفاده از تابع داخلی set

python example - sets.py

```
1 # Sets in Python (در پایتون)  
2 # Create Set (روش ایجاد مجموعه)  
3  
4 # First Meyhod: {}  
5 set_example_1 = {"1", "2", 1, 2, 3, 4}  
6  
7 print(f"First Way is {set_example_1}")  
8 print(f"type of set_example_1: {type(set_example_1)}\n\n")  
9  
10 # Second Method:  
11 set_example_2 = set(("1", "2", 1, 2, 3, 4))  
12  
13 print(f"Second Way is {set_example_2}")  
14 print(f"type of set_example_2: {type(set_example_2)}")
```

شکل ۴-۵-۴ مثالی از روش‌های تعریف مجموعه در پایتون

:۵-۴ شکل ۴- خروجی

First Way is {1, 2, 3, '2', '1', 4}

type of set_example_1: <class 'set'>

Second Way is {1, 2, '1', '2', 3, 4}

type of set_example_2: <class 'set'>

داده‌هایی که می‌توان در داخل مجموعه‌ها استفاده کرد اعم از:

۱. اعداد: داده‌هایی مانند اعداد صحیح یا اعشاری را می‌توان در مجموعه قرار داد.
۲. رشته‌ها: رشته‌ها غیرقابل تغییر هستند و می‌توانند در مجموعه‌ها ذخیره شوند.
۳. تاپل‌ها: از آنجا که تاپل‌ها غیرقابل تغییر هستند، می‌توانند عضو یک مجموعه باشند. اما تاپل‌هایی که شامل داده‌های قابل تغییر باشند (مانند لیست‌ها) نمی‌توانند در مجموعه قرار گیرند.
۴. مقادیر بولی: مقادیر True و False نیز می‌توانند در مجموعه ذخیره شوند.

python example - sets.py

```
18 # Sets in Python (مجموعه‌ها در پایتون)  
19 # داده‌هایی که می‌توان در داخل مجموعه‌ها استفاده کرد  
20  
21 set_example_3 = {"Reza", "Red", "Corvette", "1", "-0.4"} # Strings in Set  
22 print(f"set_example_3: {set_example_3} and Type of set_example_3 is {type(set_example_3)}")  
23  
24 set_example_4 = {1, 2, 6, 8, 0.5, -523, 510.249, -7.8} # Numbers in Set  
25 print(f"set_example_4: {set_example_4} and Type of set_example_4 is {type(set_example_4)}")  
26  
27 set_example_5 = {True, False} # Boolean in Set  
28 print(f"set_example_5: {set_example_5} and Type of set_example_5 is {type(set_example_5)}")  
29  
30 set_example_6 = {"Reza", False, 23, 188.8, "Yellow", "1"} #  
31 print(f"set_example_6: {set_example_6} and Type of set_example_6 is {type(set_example_6)}")
```

شکل ۴-۵۵- نوع داده‌های مجاز برای استفاده در داخل مجموعه‌ها

set_example_3: {'Red', '-0.4', 'Corvette', 'Reza', 'I'} and Type of set_example_3 is

<class 'set'>

set_example_4: {0.5, 1, 2, 6, 8, -523, -7.8, 510.249} and Type of set_example_4 is <class

'set'>

set_example_5: {False, True} and Type of set_example_5 is <class 'set'>

set_example_6: {False, 'Yellow', 23, 'Reza', 188.8, 'I'} and Type of set_example_6 is

<class 'set'>

۴-۲-۶-۲) تعداد عناصر موجود در مجموعه‌ها

برای به دست آوردن تعداد عناصر یک مجموعه می‌توان از تابع len() همانند شکل ۴-۵۶ استفاده کرد.

python example - sets.py

```
35 # Sets in Python (مجموعه‌ها در پایتون)  
36 # Length of Sets آوردن اعداد عناصر مجموعه  
37  
38 set_example_7 = {1, 2, 6, 8, 0.5, -523, 510.249, -7.8}  
39 print(f"length of {set_example_7} is {len(set_example_7)}")  
40  
41 set_example_8 = {}  
42 print(f"length of {set_example_8} is {len(set_example_8)}")  
43  
44 set_example_9 = {"Reza", False, 23, 188.8, "Yellow", "1"}  
45 print(f"length of {set_example_9} is {len(set_example_9)}")  
46  
47 set_example_10 = {True}  
48 print(f"length of {set_example_10} is {len(set_example_10)}")
```

شکل ۴-۵۶- مثالی از به دست آوردن تعداد عناصر یک مجموعه با استفاده از تابع *len*

خروجی شکل ۴-۵۶:

length of {0.5, 1, 2, 6, 8, -523, -7.8, 510.249} is 8

length of {} is 0

length of {False, 'Yellow', 'Reza', 23, '1', 188.8} is 6

length of {True} is 1

۴-۲-۶-۳) توابع وابسته‌ی مربوط به مجموعه‌ها در پایتون

جدول ۴-۶- جدول مربوط به توابع وابسته موجود در نوع داده‌ی مجموعه

تابع وابسته	توضیحات	مثال
add(element)	برای اضافه کردن یک عنصر به مجموعه استفاده می‌شود.	mySet = {'a', 'b', 'c'} mySet.add('d') print(mySet) # {'a', 'b', 'c', 'd'}
clear()	تمامی عناصر داخل مجموعه را حذف می‌کند. مقداری نمی‌گیرد.	mySet = {'a', 'b', 'c'} mySet.clear() print(mySet) # {}
copy()	از مجموعه کپی‌گیری کرده و مقدار کپی شده را بر می‌گرداند.	mySet = {'a', 'b', 'c'} newSet = mySet.copy() print(newSet) # {'a', 'b', 'c'}
discard(element)	بازوجه به مقداری که می‌گیرد، آن را در مجموعه جستجو می‌کند و اگر پیدا کند، آن را حذف می‌کند. در غیر این صورت هیچ اتفاقی نمی‌افتد. یک مقدار می‌گیرد.	mySet = {'a', 'b', 'c'} mySet.discard('a') print(mySet) # {'b', 'c'}
pop()	یک مقداری را از مجموعه حذف می‌کند. مقداری نمی‌گیرد.	mySet = {'a', 'b', 'c', 'd', 'e', 'f'} mySet.pop() print(mySet) # {'a', 'b', 'c', 'e', 'f'}

mySet = {'a', 'b', 'c'}	باتوجه به مقداری که می‌گیرد، آن را در مجموعه
mySet.discard('c')	جستجو می‌کند و اگر پیدا کند، آن را حذف می‌کند.
print(mySet)	در غیر این صورت خطأ (Error) می‌دهد.
# {'a', 'b'}	یک مقدار می‌گیرد.

remove(item)

	تفاضل مجموعه‌ها یا همان علامت تفریق (-)
	می‌باشد. بین دو یا چند مجموعه انجام می‌گیرد.
mainSet = {'a', 'b', 'c'}	مقداری که بر می‌گرداند شامل عناصری است که
set1 = {'c', 'd', 'e'}	فقط در مجموعه اول وجود دارد و در مجموعه‌های
result = mainSet.difference(difference(set1, set2,
set1)	set3, ...)
print(result)	مقدار برگردانده شده در یک مجموعه جدید قرار
# {'a', 'b'}	می‌گیرد.
	یک یا چندین مجموعه می‌گیرد.

همان علامت = - می‌باشد. بین دو یا چند مجموعه

mainSet = {'a', 'b', 'c'}	انجام می‌گیرد. مقداری که بر می‌گرداند شامل
set1 = {'c', 'd', 'e'}	عناصری است که فقط در مجموعه اول وجود دارد و
mainSet.difference_update(set	difference_update(set1,
1)	در مجموعه‌های دیگر وجود ندارد.
print(mainSet)	مقدار برگردانده شده در مجموعه اولیه قرار می‌گیرد.
# {'a', 'b'}	یک یا چندین مجموعه می‌گیرد.

set2, set3, ...)

mainSet = {'a', 'b', 'c'}	نقش & را بازی می‌کند و عناصری بر می‌گرداند که
set1 = {'c', 'd', 'e', 'a'}	در دو یا چندین مجموعه مشترک هستند (اشتراک
result =	مجموعه‌ها در ریاضیات). توجه داشته باشید که آن
mainSet.Intersection(set1)	مقادیر را در یک مجموعه جداگانه قرار می‌دهد.
print(result)	یک یا چندین مجموعه می‌گیرد.
# {'a', 'c'}	

intersection(set1, set2,
 set3, ...)

mainSet = {'a', 'b', 'c'}	نقش <code>= &</code> را بازی می‌کند و عناصری برمی‌گرداند که در دو یا چندین مجموعه مشترک هستند. توجه داشته باشید که آن مقادیر را در یک مجموعه اولیه قرار می‌دهد.	intersection_update(set1, set2, set3, ...)
set1 = {'c', 'd', 'e', 'a'}		
mainSet.intersection(set1)		
print(mainSet)		
# {'a', 'c'}	یک یا چندین مجموعه می‌گیرد.	
result = mainSet.isdisjoint(set1)	مقدار True یا False برمی‌گرداند و بررسی می‌کند که آیا دو مجموعه عناصر مشترکی باهم دارند یا نه.	isdisjoint(set2)
print(result)	اگر حداقل یک عنصر مشترک داشته باشد، مقدار True برمی‌گرداند.	
# True	یک مقدار می‌گیرد.	
result = mainSet.issubset(set1)	مقدار True یا False برمی‌گرداند و بررسی می‌کند که آیا تمامی عناصر مجموعه اول(اصلی) در داخل مجموعه دوم موجود می‌باشد یا خیر.	issubset(set2)
print(result)	یک مقدار می‌گیرد و آن هم مجموعه دوم.	
# True		
result = mainSet.issuperset(set1)	مقدار True یا False برمی‌گرداند و بررسی می‌کند که آیا تمامی عناصر مجموعه دوم در داخل مجموعه اول(اصلی) موجود می‌باشد یا خیر.	issuperset(set2)
print(result)	یک مقدار می‌گیرد و آن هم مجموعه دوم.	
# False		

```

Set1 = {'apple', 'orange',
        'cherry'}

Set2 = {'samsung', 'xiaomi',
        'apple'}

Result = Set1.symmetric_difference(Set2)
print(result)

# {'orange', 'cherry',
  'samsung', 'xiaomi'}

```

از این تابع وابسته برای برای تفاضل مجموعه‌ها استفاده می‌شود.

نتیجه را در یک مجموعه جدید قرار می‌دهد. یک مقدار می‌گیرد.

symmetric_difference
(set2)

```

Set1 = {'apple', 'orange',
        'cherry'}

Set2 = {'samsung', 'xiaomi',
        'apple'}

Set1.symmetric_difference_update(Set2)
print(Set1)

# {'orange', 'cherry',
  'samsung', 'xiaomi'}

```

از این تابع وابسته برای برای تفاضل مجموعه‌ها استفاده می‌شود.

نتیجه را با عناصر مجموعه اصلی جایگزین می‌کند. یک مقدار می‌گیرد.

symmetric_difference_update(set2)

```

Set1 = {'apple', 'orange',
        'cherry'}

Set2 = {'samsung', 'xiaomi',
        'apple'}

Result = Set1.union(Set2)
print(result)

# {'apple', 'orange', 'cherry',
  'samsung', 'xiaomi', 'apple'}

```

به اتحاد مجموعه‌ها در ریاضیات گفته می‌شود. به عبارت دیگر برای ادغام دو یا چند مجموعه بکار می‌رود. نتیجه را در یک مجموعه جدید قرار می‌دهد. یک یا چند مقدار (مجموعه) می‌گیرد.

union(set1, set2, set3,
...)

```

Set1 = {'apple', 'orange',
        'cherry'}
Set2 = {'samsung', 'xiaomi',
        'apple'}
Set1.update(Set2)
print(Set1)
# {'apple', 'orange', 'cherry',
  'samsung', 'xiaomi', 'apple'}

```

برای ادغام دو یا چند مجموعه استفاده می‌شود.
نتیجه را در مجموعه اصلی جایگزین می‌کند. update(itable)
یک یا چند مقدار (مجموعه) می‌گیرد.

۴-۶-۲-۴) دسترسی به عناصر مجموعه

باتوجه به نکات قبلی، امکان دسترسی مستقیم به عناصر مجموعه‌ها با اندیس و یا توابع وابسته وجود ندارد اما می‌توان با استفاده از حلقه for و یا استفاده از کلیدواژه in از وجود یا عدم وجود عنصر مورد نظر درون مجموعه اطمینان حاصل کرد. در ادامه برای درک این موضوع به شکل ۵۷-۴ توجه کنید.

python example - sets.py

```

53 # Sets in Python (مجموعه‌ها در پایتون)
54 # Is Exist or NOT ??? ؟؟؟
55
56 set_example_11 = {"Reza", False, 23, 188.8, "Yellow", "1"}
57
58 print(f"Is 'Blue' in set_example_11: {'Blue' in set_example_11}")
59
60 print(f"Is '1' in set_example_11: {'1' in set_example_11}")
61
62 print(f"Is 'True' in set_example_11: {True in set_example_11}")
63
64 print(f"Is 'Reza' in set_example_11: {'Reza' in set_example_11}")
65
66 print(f"Is 23 in set_example_11: {23 in set_example_11}")

```

شکل ۴-۵۷-۴-مثالی از وجود داشتن یا نداشتن یک عنصر در داخل یک مجموعه

Is 'Blue' in set_example_11: False

Is '1' in set_example_11: True

Is 'True' in set_example_11: False

Is 'Reza' in set_example_11: True

Is 23 in set_example_11: True

۴-۲-۶) افزودن عنصر/عناصر جدید به مجموعه‌ها

برای افزودن فقط یک عنصر به یک مجموعه، می‌توان از تابع وابسته `add` استفاده کرد و برای افزودن چند عنصر به یک مجموعه می‌توان از تابع وابسته `update` استفاده کرد. در ادامه به شکل ۴-۵۸ توجه کنید.

نکته: مقداری که تابع وابسته `update` می‌گیرد می‌تواند شامل هر نوع داده‌ای که قابلیت پیمایش دارد، باشد.

python example - sets.py

```
70 # Sets in Python (مجموعه‌ها در پایتون)  
71 # Add element/elements into Set افزودن یک یا چند عنصر به مجموعه  
72  
73 set_example_12 = {"Reza"}  
74 print(f"Before Add: {set_example_12}")  
75  
76 # Add one element  
77 set_example_12.add("Mohammad")  
78 print(f"After add one Element: {set_example_12}")  
79  
80 # Add many elements  
81 set_example_12.update(['Abolfazl', 'Mahdi', 'Ali'])  
82 print(f"After add many elements: {set_example_12}")
```

شکل ۴-۵۸-۱- مثالی از افزودن یک یا چند عنصر به داخل یک مجموعه

خروجی شکل ۴-۵۸:

Before Add: {'Reza'}

After add one Element: {'Mohammad', 'Reza'}

After add many elements: {'Mahdi', 'Ali', 'Reza', 'Abolfazl', 'Mohammad'}

۴-۲-۶) حذف عنصر از مجموعه یا حذف تمامی عناصر از مجموعه

معمولًا برای حذف یک عنصر مشخص از توابع وابسته‌ی remove و discard طبق شکل ۴-۵۹ استفاده می‌کنند.

تفاوتی که توابع وابسته‌ی `remove` و `discard` دارند در این است که اگر آن عنصر درون مجموعه وجود نداشته باشد، تابع وابسته `remove` ارور و خطای می‌دهد و برنامه متوقف خواهد شد اما، تابع وابسته `discard` در صورت نبود عنصر درون مجموعه، هیچ خطایی نخواهد داد و برنامه متوقف نخواهد شد.

شما می‌توانید از تابع وابسته `pop` نیز استفاده کنید اما این تابع وابسته به صورت تصادفی و شансی یک عنصر از مجموعه را حذف خواهد کرد و شما نمی‌توانید یک عنصر مشخص را با تابع وابسته `pop` حذف کنید.

python example - sets.py

```
86 # Sets in Python (مجموعه‌ها در پایتون)  
87 # Remove one element from Set  
88  
89 set_example_13 = {'Abolfazl', 'Reza', 'Ali', 'Mahdi', 'Mohammad', 'Hossein', 'Amir'}  
90 # print(f"Before Removing: {set_example_13}")  
91  
92 # Remove by discard method  
93 set_example_13.discard('Ali')  
94 print(f"Remove 'Ali' by discard method: {set_example_13}")  
95  
96 # Remove by remove method  
97 set_example_13.remove('Hossein')  
98 print(f"Remove 'Hossein' by remove method: {set_example_13}")  
99  
100 # Remove by pop method  
101 set_example_13.pop()  
102 print(f"Remove Random element by pop method: {set_example_13}")
```

شكل ۴-۵۹- مثالی از حذف یک عنصر از مجموعه با استفاده از توابع وابسته‌ی `remove` و `discard` و `pop`

خروجی شکل ۴-۵۹:

Remove 'Ali' by discard method: {'Abolfazl', 'Hossein', 'Reza', 'Mahdi', 'Amir', 'Mohammad'}

Remove 'Hossein' by remove method: {'Abolfazl', 'Reza', 'Mahdi', 'Amir', 'Mohammad'}

Remove Random element by pop method: {'Reza', 'Mahdi', 'Amir', 'Mohammad'}

معمولاً برای حذف تمامی عناصر یک مجموعه از تابع وابسته `clear` استفاده می‌شود و نتیجه، یک مجموعه خالی و فاقد عنصر همانند شکل ۴-۶۰ خواهد بود.

python example - sets.py

```
106 # Sets in Python (مجموعه‌ها در پایتون)  
107 # Remove all elements from Set مجموعه  
108  
109 # remove by clear method  
110 set_example_14 = {'Abolfazl', 'Reza', 'Ali', 'Mahdi', 'Mohammad', 'Hossein', 'Amir'}  
111 print(f"Before Removing: {set_example_14}")  
112  
113 set_example_14.clear()  
114 print(f"After Removing: {set_example_14}")
```

شکل ۴-۶۰- مثالی از حذف تمامی عناصر مجموعه با استفاده از تابع وابسته `clear`

خروجی شکل ۴-۶۰:

Before Removing: {'Mohammad', 'Ali', 'Hossein', 'Abolfazl', 'Amir', 'Mahdi', 'Reza'}

After Removing: set()

کلید واژه `del` نیز تمامی مجموعه را پاک خواهد کرد اما زمانی که شما، نام آن مجموعه را فرا خوانید، خطای دریافت خواهید کرد.

۴-۲-۶-۷) ادغام دو یا چند مجموعه

برای ادغام دو یا چند مجموعه روش‌های مختلفی وجود دارد:

- توابع وابسته `union` و `update` (شکل ۴-۶۱).
- تابع وابسته `intersection` (فقط مقادیر تکراری را برمی‌گرداند) (شکل ۴-۶۲).
- تابع وابسته `difference` (مقادیری که در مجموعه اول بود و در مجموعه‌های دیگر نبوده را برمی‌گرداند) (شکل ۴-۶۲).
- تابع وابسته `symmetric_difference` (همه مقادیر را برمی‌گرداند به جز مقادیر تکراری) (شکل ۴-۶۲).

python example - sets.py

```
118 # Sets in Python (مجموعه‌ها در پایتون)  
119  
120 # union and update methods  
121 set_example_15 = {'Abolfazl', 'Reza'}  
122 adding_set_1 = set(['Ali', 'Mahdi', 'Mohammad'])  
123 adding_set_2 = {'Hossein', 'Amir'}  
124 print(f"Before adding: {set_example_15}\n")  
125  
126 # union Method  
127 myNewSet = set_example_15.union(adding_set_1)  
128 print(f"adding by union method: {myNewSet}\n")  
129  
130 # update method  
131 set_example_15.update(adding_set_2)  
132 print(f"adding by update method: {set_example_15}")
```

شکل ۴-۶۱- مثالی از استفاده از توابع وابسته `union` و `update`

Before adding: {'Reza', 'Abolfazl'}

adding by union method: {'Reza', 'Abolfazl', 'Ali', 'Mahdi', 'Mohammad'}

adding by update method: {'Reza', 'Hossein', 'Amir', 'Abolfazl'}

python example - sets.py

```
136 # Sets in Python (مجموعه‌ها در پایتون)
137
138 # intersection and difference and symmetric_difference methods
139 set_example_16 = {'Abolfazl', 'Reza', 'Ali', 'Mahdi', 'Mohammad'}
140 set_example_17 = {'Hossein', 'Amir', 'Mahdi', 'Reza', 'Ali'}
141 print(f'Default Sets:\n\t set Example 16: {set_example_16}\n\t set Example 17: {set_example_17}\n')
142 # intersection method
143 intersection_result = set_example_16.intersection(set_example_17)
144 print(f'intersection of set_example_16 and set_example_17 is {intersection_result}\n')
145
146 # difference method
147 difference_result = set_example_16.difference(set_example_17)
148 print(f'difference of set_example_16 and set_example_17 is {difference_result}\n')
149
150 # symmetric_difference methods
151 symmetric_difference_result = set_example_16.symmetric_difference(set_example_17)
152 print(f"symmetric_difference of set_example_16 and set_example_17 is {symmetric_difference_result}")
```

شکل ۶۲-۴- استفاده از توابع وابسته‌ی *intersection* و *symmetric_difference*

Default Sets:

set Example 16: {'Mahdi', 'Reza', 'Mohammad', 'Ali', 'Abolfazl'}

set Example 17: {'Mahdi', 'Reza', 'Hossein', 'Ali', 'Amir'}

intersection of set_example_16 and set_example_17 is {'Mahdi', 'Ali', 'Reza'}

difference of set_example_16 and set_example_17 is {'Abolfazl', 'Mohammad'}

symmetric_difference of set_example_16 and set_example_17 is {'Hossein', 'Mohammad', 'Amir', 'Abolfazl'}

نکات:

✓ می‌توانید به جای تابع وابسته union از علامت | برای ادغام دو یا چند مجموعه طبق شکل ۶۳-۴ استفاده کنید. البته توجه داشته باشید که زمانی که از این علامت استفاده می‌کنید، باید تمامی داده‌ها از نوع set باشند.

✓ می‌توانید به جای تابع وابسته intersection از علامت & بین دو یا چندین مجموعه طبق شکل ۶۳-۴ استفاده کنید. برای استفاده از این علامت نیز باید تمامی داده‌ها از نوع set باشند.

✓ می‌توانید به جای تابع وابسته difference از علامت منها (-) بین دو یا چندین مجموعه استفاده کنید. برای استفاده از این علامت نیز باید تمامی داده‌ها از نوع set باشند. برای درک بهتر به شکل ۶۴-۴ توجه کنید.

✓ می‌توانید به جای تابع وابسته `symmetric_difference` از علامت `^` بین دو یا چندین مجموعه طبق

شکل ۴-۶۴ استفاده کنید. برای استفاده از این علامت نیز باید تمامی داده‌ها از نوع `set` باشند.

✓ مقادیر ۱ و `True` یکسان هستند و همچنین مقادیر ۰ و `False` بنا بر این زمانی که در یک مجموعه عناصر

۱ و `True` وجود داشته باشد، ۱ حذف خواهد شد چرا که با `True` یکسان می‌باشد. برای ۰ و `False` نیز

این مورد صحت دارد. به شکل ۴-۶۵ توجه کنید

python example - sets.py

```
156 # Sets in Python (مجموعه‌ها در پایتون)  
157  
158 set_example_18 = {'Abolfazl', 'Reza'}  
159 set_example_19 = {'Mahdi', 'Reza', 'Ali'}  
160 set_example_20 = {'Reza', 'Hossein', 'Amir'}  
161 set_example_21 = {'Hossein', 'Reza', 'Ali'}  
162  
163 # union = |  
164 union_result = set_example_18 | set_example_19 | set_example_20 | set_example_21  
165 print(f"Result of union is {union_result}")  
166  
167 # intersection = &  
168 intersection_result = set_example_18 & set_example_19 & set_example_20 & set_example_21  
169 print(f"Result of intersection is {intersection_result}")
```

شکل ۴-۶۳-۶۴- مثال مربوط به نکات توابع وابسته‌ی `union` و `intersection`

خروجی شکل ۴-۶۳:

Result of union is {'Ali', 'Abolfazl', 'Hossein', 'Mahdi', 'Amir', 'Reza'}

Result of intersection is {'Reza'}

python example - sets.py

```
173 # Sets in Python (مجموعه‌ها در پایتون)  
174  
175 set_example_18 = {'Abolfazl', 'Reza'}  
176 set_example_19 = {'Mahdi', 'Reza', 'Ali'}  
177 set_example_20 = {'Reza', 'Hossein', 'Amir'}  
178 set_example_21 = {'Hossein', 'Reza', 'Ali'}  
179  
180 # difference = -  
181 difference_result = set_example_18 - set_example_19 - set_example_20 - set_example_21  
182 print(f"Result of difference is {difference_result}")  
183  
184 # symmetric_difference = ^  
185 symmetric_difference_result = set_example_18 ^ set_example_19 ^ set_example_20 ^ set_example_21  
186 print(f"Result of symmetric_difference is {symmetric_difference_result}")
```

شکل ۴-۶۴- مثال مربوط به نکات توابع وابسته‌ی *difference* و *symmetric_difference*

خروجی شکل ۶۴-۴:

Result of difference is {'Abolfazl'}

Result of symmetric_difference is {'Mahdi', 'Abolfazl', 'Amir'}

python example - sets.py

```
190 # Sets in Python (مجموعه‌ها در پایتون)  
191  
192 # True = 1  
193 set_example_22 = {True, 1}  
194 print(f"Result: {set_example_22}")  
195  
196 # False = 0  
197 set_example_23 = {False, 0}  
198 print(f"Result: {set_example_23}")  
199  
200 # Combination  
201 set_example_24 = {1, False, 0, True}  
202 print(f"Result: {set_example_24}")
```

شکل ۴-۶۵- مثال مربوط به نکات مقادیر *False* و *True* در داخل مجموعه‌ها

خروجی شکل ۴-۶۵:

Result: {True}

Result: {False}

Result: {False, 1}

فصل پنجم

دستورات شرطی در زبان برنامه‌نویسی پایتون

۵) دستورات شرطی در پایتون

عبارات شرطی، دستوراتی هستند که برای تصمیم‌گیری در شرایط خاص استفاده می‌شوند. به عبارت دیگر، برنامه برای شرایط خاص، دستوراتی که به صورت کد تعریف شده است، اجرا خواهد کرد. معمولاً این دستورات با توجه به نتیجه شرطی (True/False) که گذاشته شده است اجرا می‌شوند. برای مثال اگر این شرط درست باشد، تصمیم اول را اجرا می‌کند اما اگر این شرط درست نباشد تصمیم دوم را اجرا خواهد کرد. برای درک بهتر این موضوع به سه مثال زیر توجه کنید.

مثال ۱: فرض کنید برنامه‌ای می‌نویسید، به جایی از کد می‌رسید که کاربر می‌تواند چند انتخاب داشته باشد و هر کدام از انتخاب‌ها، نتایجی منحصر به‌فردی دارد، در این صورت برای این‌که برای هر انتخاب یک نتیجه‌ای در کدتان بنویسید باید از دستورات شرطی استفاده کنید.

مثال ۲: فرض کنید یک فهرستی از میوه‌های مختلف را در کدهایتان نوشته‌اید، و هم‌چنین در کدتان از کاربر خواسته‌اید تا میوه مورد علاقه‌ی خود را بنویسید، و شما می‌خواهید بدانید که آیا میوه‌ای که کاربر نوشته در داخل فهرست میوه‌هایی که قبلاً در کدتان نوشته‌اید وجود دارد یا خیر. اگر وجود داشته باشد، به کاربر اطلاع دهد که این میوه در فهرست میوه‌ها قرار دارد و اگر وجود نداشته باشد، از کاربر عذرخواهی کرده و عدم وجود میوه‌ی مورد علاقه‌ی کاربر در فهرست میوه‌ها را گزارش کند.

مثال ۳: در مثال سوم کدی باید بنویسید که از کاربر عددی دریافت کند و زوج یا فرد بودن آن عدد را در خروجی چاپ کند. اگر باقی‌مانده تقسیم آن عدد بر ۲ برابر صفر باشد، آن عدد زوج است و اگر باقی‌مانده تقسیم آن عدد بر ۲ برابر یک باشد، آن عدد فرد می‌باشد. (باقی‌مانده تقسیم یک عدد بر عدد دیگر را با علامت $\%$ نشان داده می‌شود). در این موضع باید از دستورات شرطی استفاده کنید.

مثال ۴: در پروژه کد نویسی ماشین حساب ساده نیز یکی از راه‌ها استفاده از دستورات شرطی است. به این صورت که از کاربر سه مقدار گرفته می‌شود، مقادیر اول و دوم، اعداد اول و دوم هستند. مقدار سوم، عملیات محاسباتی

مثل جمع، تفریق، ضرب و تقسیم است که بین دو عدد اعمال می‌شود. حال باید برای هر کدام از عملیات‌ها یک دستور نوشته شود. برای مثال اگر مقدار سومی که کاربر وارد کرده، "+" باشد، دو عدد را باهم جمع کند، در غیر این صورت اگر مقدار سوم برابر "-" باشد، عدد اول منهای عدد دوم شود، در غیر این صورت اگر مقدار سوم برابر "*/" باشد، عدد اول را ضرب در عدد دوم کند، در غیر این صورت اگر مقدار سوم برابر "/" باشد، عدد اول را تقسیم بر عدد دوم کند و اگر هیچ‌کدام از موارد بالا نباشد، کاربر، پیام خطأ دریافت کند.

۱-۵) دستور شرطی if

یکی از بنیادی‌ترین و پراستفاده‌ترین ساختارهای کنترلی در برنامه‌نویسی، دستور شرطی if است. این دستور در پایتون برای تصمیم‌گیری و اجرای کد بر اساس شرایط خاص به کار می‌رود. به کمک if می‌توانید تعیین کنید که کدام بخش از کد در صورت صحیح بودن یک شرط (یا مجموعه‌ای از شرایط) اجرا شود. دستور شرطی if به برنامه نویسان اجازه می‌دهد تا روی یک دنباله از عناصر مانند فهرست‌ها، تاپل‌ها، رشته‌ها یا ... پیمايش کرده و تکرار شوند و یک بلوک کد را برای هر عنصر اجرا کنند. این ساختار به ویژه برای کارهایی مانند پیمايش مجموعه‌ها، پردازش داده‌ها یا تولید خروجی‌های تکراری مفید است.

ساختار کلی دستور if در پایتون به صورت زیر است:

مشرط if

دستورات

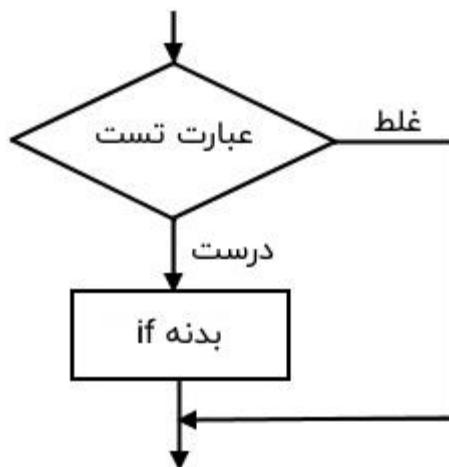
اگر شرط نوشته شده در بخش if ارزیابی شود و نتیجه‌ی آن True باشد، کدهای داخل بلاک if اجرا می‌شوند. در غیر این صورت، این دستورات نادیده گرفته می‌شوند.

نکته: پایتون یک زبان حساس به دندانه‌گذاری^{۵۰} است و استفاده صحیح از آن برای ساختاردهی کد الزامی است. برخلاف برخی زبان‌های دیگر که از آکولاد ({}) یا کلمات کلیدی مشخصی برای مشخص کردن بلوک‌های کد استفاده می‌کنند، پایتون دندانه‌گذاری را به عنوان بخشی از ساختار استفاده می‌کند. در دستورات شرطی if، دندانه‌گذاری برای مشخص کردن بلوک کدی که باید در صورت برقرار بودن شرط اجرا شود، ضروری است.

قوانين دندانه‌گذاری در پایتون

- استفاده از فضای خالی یا تب: هر خط کدی که متعلق به یک بلوک خاص (مانند بلوک if یا for یا while) باشد، باید دندانه‌گذاری شود.

- یکسان بودن دندانه‌گذاری: تمامی خطوط داخل یک بلوک باید دندانه‌گذاری یکسانی داشته باشند (مثلاً فاصله ۴ تاب).^۱



شکل ۵-۱- فلوچارت مربوط به دستورات شرطی:

ساختار دستورات شرطی:

- ساختار اول: ساختاری است که فقط با if تشکیل شده است.

⁵⁰ Indentation

- ساختار دوم: ساختاری است که با `if` و `else` تشکیل شده است.
- ساختار سوم: ساختاری است که با `if`, `elif` و `else` تشکیل شده است.

۱-۱-۵) ساختار `if`

این ساختار با کلیدواژه `if` شروع می‌شود و در ادامه‌ی آن یک شرط^{۵۱} می‌آید و سپس بعد از شرط، علامت دو نقطه روی هم (:) نوشته می‌شود. و سپس در خط بعدی ابتدا دندانه‌گذاری و سپس دستورالعمل‌ها^{۵۲} به صورت کد نوشته می‌شود. به شکل ۲-۵ توجه نمایید.

اغلب زبان‌های برنامه‌نویسی مانند سی، سی‌پلاس‌پلاس و جاوا از آکولاد برای تعریف کردن یک بلوک از کد استفاده می‌کنند. در پایتون، برای انجام این کار، از دندانه‌گذاری استفاده می‌شود. یک بلوک کد (دستورات شرطی، حلقه‌ها، بدنی یک تابع و کلاس‌ها) با دندانه‌گذاری آغاز می‌شود و با اولین خط که فاقد تورفتگی است، پایان می‌یابد. نکته: اگر دندانه‌گذاری در دستورات `if` وجود نداشته باشد، برنامه اجرا نخواهد شد و خطا خواهد داد.

ساختار کلی:

if condition:

statement

^{۵۱} Condition

^{۵۲} Statement

python example - condition_Statement.py

```
1 # Condition Statement in Python (دستورات شرطی در پایتون)
2
3 # First Syntax (Only if) ساختار اول: تعریف دستورات شرطی فقط با if
4 """
5     if condition:
6         statement
7 """
8
9 x = 25
10 if x%2 == 1:
11     print(f"x:{x} is Odd number")
12
13 name = "Reza"
14 if name == "ali":
15     print("Hello Ali")
```

شکل ۵-۲- مثالی از تعریف عبارات شرطی با `if`

خروجی شکل ۵-۲:

x:25 is Odd number

۲-۱-۵ ساختار `if/else`

قسمت اول این ساختار دقیقا همانند ساختار اول می باشد، اما در قسمت کلیدواژه `else` به ساختار اضافه می شود.
به این صورت که ساختار اول را نوشته و دستورالعمل های `if` را نوشته و سپس در خط بعدی بدون دندانه گذاری،
کلیدواژه `else` را نوشته و سپس بعد از کلیدواژه `else` دو نقطه روی هم گذاشته می شود (برای کلیدواژه
شرط گذاشته نمی شود). در نهایت بعد از گذاشتن دونقطه روی هم، خط بعدی، ابتدا دندانه گذاری می شود و سپس

دستورالعمل‌های مربوط به `else` نوشته می‌شود. منظور از `if` درست نباشد، دستورات `else` اجرا شود. ساختار کلی:

if condition:

statement

else:

statement

python example - condition_Statement.py

(دستورات شرطی در پایتون)
ساختار دوم: تعریف دستورات شرطی با `if/else`

```
19 # Condition Statement in Python
20
21 # Second Syntax (if/else) with condition
22 """
23 if condition:
24     statement
25 else:
26     statement
27 """
28
29 random_number = 1254
30 if random_number%2 == 0:
31     print(f"{random_number} is Even number.")
32 else:
33     print(f"{random_number} is Odd number.")
34
35 user_score = 5
36 # if user score grater than 10, user pass else Reject
37 if user_score > 10:
38     print("Pass.")
39 else:
40     print("Reject.")
```

شكل ۵-۳- مثالی از تعریف عبارات شرطی با `if/else`

1254 is Even number.

Reject.

۱-۳) ساختار if/elif/else

این ساختار از سه قسمت تشکیل شده است و معمولاً از این ساختار زمانی استفاده شود که اگر شرط اول رد شود، شرط دوم را بررسی کند، اگر شرط دوم رد شود شرط سوم را بررسی کند و ... و در نهایت اگر همهی شرط‌ها رد شوند، دستورالعمل‌هایی مربوط به else اجرا خواهد شد. به بیانی دیگر اگر شما چندین شرط داشته باشید، با کلیدواژه elif می‌توانید آن شرط‌ها را بررسی کنید. دستورالعمل‌های مربوط به elif نیز باید با دندانه‌گذاری نوشته شوند.

قسمت اول این ساختار، همانند ساختار اول می‌باشد که در بالا توضیح داده شد. قسمت دوم با کلیدواژه elif شروع می‌شود. به این صورت نوشته می‌شود که بعد از دستورالعمل‌های if، در خط جدید، کلیدواژه elif (بدون دندانه‌گذاری) نوشته می‌شود و در ادامه شرط مربوط به elif نوشته می‌شود و با دو نقطه روی هم به خط بعدی رفته و با دندانه‌گذاری، دستورالعمل‌های مربوط به elif نوشته می‌شود. قسمت سوم مربوط به کلیدواژه else می‌باشد که همانند ساختار دوم نوشته می‌شود.

نکته: ممکن است یک برنامه دو شرط داشته باشد مانند شکل ۴-۵، که در این صورت نیاز به یک if و elif برای شرط‌های نیاز است. اما ممکن است یک برنامه بیش از دو شرط داشته باشد مانند شکل ۵-۵، که در این صورت باید برای هر شرط یک elif گذاشته شود. ساختار کلی:

۱-۳-۱-۵) دستورات شرطی if/elif/else با دو شرط

if condition_1:

statement

elif condition_2:

statement

else:

statement

python example - condition_Statement.py

```
44 # Condition Statement in Python (دستورات شرطی در پایتون)  
45  
46 # Third Syntax (if/elif/else) با ساختار سوم: تعريف دستورات شرطی (if/elif/else)  
47 # Two Condition دستورات شرطی با دو شرط  
48  
49 user_score = 17  
50  
51 if user_score >= 17:  
52     print("Good Job.")  
53 elif 10 < user_score < 17:  
54     print("Not Bad.")  
55 else:  
56     print("It's Bad.")
```

شکل ۴-۵- مثالی از تعريف عبارات شرطی if/elif/else با دو شرط

خروجی شکل ۴-۵:

Good Job.

۵-۱-۳-۲) ساختار **if/elif/else** با بیش از دو شرط

if condition_1:

statement

elif condition_2:

statement

elif condition_3:

statement

...

else:

statement

python example - condition_Statement.py

```
60 # Condition Statement in Python (دستورات شرطی در پایتون)  
61  
62 # Third Syntax (if/elif/else) ساختار سوم: تعریف دستورات شرطی با  
63 # Grater than Two Condition دستورات شرطی با بیش از دو شرط  
64  
65 user_color = "yellow"  
66  
67 if user_color == "green":  
68     print("First try: User color is green.")  
69 elif user_color == "red":  
70     print("Second try: User color is red.")  
71 elif user_color == "black":  
72     print("Third try: User color is black.")  
73 elif user_color == "white":  
74     print("Fourth try: User color is white.")  
75 elif user_color == "yellow":  
76     print("Fifth try: User color is yellow.")  
77 elif user_color == "blue":  
78     print("Last try: User color is blue.")  
79 else:  
80     print("Oh no, I can't guess color! :(")
```

شكل ۵-۵- مثالی از تعریف عبارات شرطی *if/elif/else* با بیش از دو شرط

خروجی شکل ۵-۵:

Fifth try: User color is yellow

۴-۱-۵) مقایسه‌ی بین شرط‌ها در دستورات شرطی

با توجه به شکل ۷۰-۴ عمل‌گرهای مقایسه‌ای که می‌توانید در دستورات شرطی بین دو یا چند شرط اعمال کنید

عبارتند از:

- $a == b$ برابر است با b •
- $a != b$ b با a برابر نیست: •
- $a < b$ a کوچکتر است از b •
- $a <= b$ a کوچکتر یا مساوی b •
- $a > b$ a بزرگتر است از b •
- $a >= b$ a بزرگتر یا مساوی b •

python example - condition_Statement.py

```

86 # Condition Statement in Python (دستورات شرطی در پایتون)
87
88 # عملگرهای مقایسه بین دو شرط
89 a = 45
90 b = 13
91 if a == b:
92     print("a and b are equal.")
93
94 if a != b:
95     print("a and b are NOT equal.")
96
97 if a > b:
98     print("a is grather than b.")
99
100 if a >= b:
101    print("a is greater than or equal to b.")
102
103 if a < b:
104    print("a is less than b.")
105
106 if a <= b:
107    print("a is less than or equal to b.")

```

شکل ۵-۶- مثال‌هایی از استفاده عملگرهای مقایسه‌ای بین دو یا چند شرط در دستورات شرطی

خروجی شکل ۵-۶:

a and b are NOT equal.

a is grather than b.

a is greater than or equal to b

کلیدواژه‌هایی که بین دو شرط طبق شکل ۷-۵ استفاده می‌شود عبارت است از:

۱. استفاده از کلیدواژه `or` بین دو شرط: اگر فقط نتیجه‌ی یکی از شرط‌ها مقدار `True` داشته باشد، دستورالعمل‌ها اجرا خواهند شد.

۲. استفاده از کلیدواژه `and` بین دو شرط: اگر نتیجه هر دو شرط `True` باشند، دستورالعمل‌ها اجرا خواهند شد.

python example - condition_Statement.py

```
111 # Condition Statement in Python (دستورات شرطی در پایتون)
112
113 # And/Or in Condition Statement
114 name = "Reza"
115 age = 23
116
117 if name == "Reza" and age == 24:
118     print("By And")
119     print(f"name = {name}:{name == 'Reza'}")
120     print("AND")
121     print(f"age = {age}: {age == 24}")
122     print(f"Result is {name == 'Reza' and age == 24}")
123
124 if name == "Reza" or age == 24:
125     print("By Or")
126     print(f"name = {name}:{name == 'Reza'}")
127     print("OR")
128     print(f"age = {age}: {age == 24}")
129     print(f"Result is {name == 'Reza' or age == 24}")
```

شكل ۷-۵-مثالی از استفاده `and` و `or` در عبارات شرطی بین دو شرط

By Or

name = 'Reza':True

OR

age = 24: False

Result is True

نکته: استفاده از کلیدواژه `in` همانند شکل ۷-۸ برای بررسی وجود داشتن یا نداشتن یک عنصر در داخل فهرست، تاپل و هر نوع داده‌ای که قابلیت پیمایش دارد.

python example - condition_Statement.py

```
133 # Condition Statement in Python (دستورات شرطی در پایتون)
134
135 # in keyword in condition statement (بررسی وجود داشتن یا نداشتن با استفاده از کلیدواژه)
136 colors_list = ["red", "green", "blue", "black", "white"]
137
138 if "yellow" in colors_list:
139     print("yellow is exist.")
140 else:
141     print("yellow is NOT exist.")
142
143
144 if 'blue' in colors_list:
145     print("blue is exist.")
146 else:
147     print("blue is NOT exist.")
```

شکل ۷-۸- مثالی از کلیدواژه `in` در داخل عبارات شرطی

yellow is NOT exist.

blue is exist.

استفاده از کلیدواژه **not** در جملات شرطی

معمولاً از این کلیدواژه جهت ضد کردن یک شرط استفاده می‌شود. برای مثال اگر عنصری در داخل فهرستی وجود نداشته باشد مانند شکل ۹-۵، دستورالعمل‌های مربوطه اجرا شود.

python example - condition_Statement.py

```
151 # Condition Statement in Python (دستورات شرطی در پایتون)  
152 # not keyword in condition statement (استفاده از کلیدواژه جهت ضد کردن جمله)  
153 colors_list = ["red", "green", "yellow", "orange", "white"]  
154  
155 if "yellow" not in colors_list:  
156     print("yellow is NOT exist.")  
157 else:  
158     print("yellow is exist.")  
159  
160  
161 if 'blue' not in colors_list:  
162     print("blue is NOT exist.")  
163 else:  
164     print("blue is exist.")
```

شکل ۹-۵- مثالی از کلیدواژه **not** در عبارات شرطی

yellow is exist.

blue is NOT exist.

۵-۱-۵) دستورات شرطی تو در تو

یکی از قابلیت‌هایی که زبان برنامه‌نویسی پایتون دارد، دستورات شرطی تو در تو^{۵۳} می‌باشد یعنی استفاده از دستورات شرطی داخل دستورات شرطی دیگر است.

برای درک بهتر این موضوع یک مثالی از خرید خودرو آورده شده است. فرض کنید می‌خواهید سن کاربر را دریافت کنید. در دستور شرطی اول، مشخص می‌کنید که اگر سن کاربر بالای هجده سال باشد، دستور شرطی دوم اجرا شود. در غیر این صورت به کاربر اطلاع دهد که زیر سن قانونی می‌باشد و امکان خرید خودرو وجود ندارد. دستور شرطی دوم، دara بودن یا نبودن گواهینامه رانندگی کاربر می‌باشد، اگر کاربر بیشتر از هجده سال داشته باشد و دارای گواهینامه رانندگی باشد، می‌تواند عملیات خرید خودرو را انجام دهد در غیر این صورت ابتدا باید گواهینامه گرفته و سپس اقدام به خرید خودرو کند. کد مثال آورده شده در شکل‌های ۱۰-۵، ۱۱-۵، ۱۲-۵ مشاهده کنید.

^{۵۳} Nested if Statement

python example - condition_Statement.py

```
168 # Condition Statement in Python (دستورات شرطی در پایتون)  
169  
170 # Nested if تو در تو دستورات شرطی تو در تو  
171 # Buy Car  
172 user_age_1 = 24  
173 user_certificate = True  
174 if user_age_1 >= 18:  
175     if user_certificate == True:  
176         print("You Can Buy Car. :")  
177     else:  
178         print("You Don't Have Certificate. So You Can't Buy Car. :(")  
179 else:  
180     print("Under legal age! :(")
```

شکل ۵-۱۰- مثال خرید خودرو برای کاربر اول با استفاده از عبارات شرطی تو در تو

خروجی شکل ۵-۱۰ :

You Can Buy Car. :)

python example - condition_Statement.py

```
184 # Condition Statement in Python (دستورات شرطی در پایتون)  
185  
186 # Nested if تو در تو دستورات شرطی تو در تو  
187 # Buy Car  
188 user_age_2 = 15  
189 user_certificate = True  
190 if user_age_2 >= 18:  
191     if user_certificate == True:  
192         print("You Can Buy Car. :)")  
193     else:  
194         print("You Don't Have Certificate. So You Can't Buy Car. :(")  
195 else:  
196     print("Under legal age! :(")
```

شكل ۱۱-۵- مثال خرید خودرو برای کاربر دوم با استفاده از عبارات شرطی تو در تو

خروجی شکل ۱۱-۵ :

Under legal age! :(

python example - condition_Statement.py

```
200 # Condition Statement in Python (دستورات شرطی در پایتون)  
201  
202 # Nested if تو در تو دستورات شرطی تو در تو  
203 # Buy Car  
204 user_age_3 = 25  
205 user_certificate = False  
206 if user_age_3 >= 18:  
207     if user_certificate == True:  
208         print("You Can Buy Car. :)")  
209     else:  
210         print("You Don't Have Certificate. So You Can't Buy Car. :(")  
211 else:  
212     print("Under legal age! :(")
```

شکل ۱۲-۵- مثال خرید خودرو برای کاربر سوم با استفاده از عبارات شرطی تو در تو

خروجی شکل ۱۲-۵ :

You Don't Have Certificate. So You Can't Buy Car. :(

نکته: هیچ‌گاه قسمت دستورالعمل‌ها را خالی نگذارید، چرا که با اجرا کردن کدها خطای دریافت خواهید کرد. اما اگر به هر دلیلی عبارات شرطی در کدهایتان وجود داشته باشد که می‌خواهید بعداً دستورالعمل‌های آن را تکمیل کنید، به جای این که بخواهید قسمت دستورالعمل‌ها را خالی بگذارید می‌توانید از کلیدواژه pass استفاده کنید. کلیدواژه pass از خطای جلوگیری می‌کند.

فصل ششم

حلقه‌ها در زبان برنامه‌نویسی پایتون

۶) حلقه‌ها در پایتون

حلقه‌های^{۵۴} از مفاهیم بنیادی و کلیدی در زبان برنامه‌نویسی پایتون به شمار می‌روند. این ساختارها با هدف اجرای عملیات تکراری طراحی شده‌اند و به برنامه‌نویسان امکان می‌دهند تا با نوشتن کدی مختصرتر و مؤثرتر، به اهداف مدنظر خود دست یابند. در زبان پایتون دو نوع حلقه اصلی وجود دارد: حلقه‌ی `for` و حلقه‌ی `while`، که هر یک دارای ساختار، فلوچارت، و الگوی اجرایی مختص به خود هستند. حلقه‌ها ابزار قدرتمندی برای ساده‌سازی فرایندهای تکراری و خودکارسازی عملیات محسوب می‌شود.

۶-۱) حلقه while در پایتون

در زبان برنامه‌نویسی پایتون، حلقه‌ی `while` یکی از ابزارهای قدرتمند برای تکرار مداوم یک بخش از کد است تا زمانی که شرط مشخصی برقرار باشد. این نوع حلقه به برنامه‌نویسان امکان می‌دهد که یک بلوک کد را پیوسته اجرا کنند، مشروط بر این‌که آن شرط هم‌چنان به عنوان "درست یا True" ارزیابی شود. به عبارتی دیگر، تا زمانی که شرط برقرار است، حلقه هم‌چنان ادامه می‌یابد؛ و زمانی که شرط نقض شود، حلقه خاتمه می‌یابد و اجرای برنامه به خط بعدی کد منتقل می‌شود.

شیوه‌ی کار با حلقه‌ی `while` به این صورت است که ابتدا شرطی تعریف می‌شود که باید حین هر بار اجرای حلقه ارزیابی شود. در صورتی که این شرط "درست یا True" باشد، بدنه‌ی حلقه اجرا خواهد شد؛ در غیر این صورت، اجرای حلقه متوقف شده و کنترل برنامه به کدی خارج از حلقه بازمی‌گردد. چنین ویژگی‌ای، کاربرد حلقه‌ی `while` را برای مواردی مانند شمارش‌های پیوسته، انجام عملیات خاص تا برآورده شدن شرطی ویژه، و حتی تکرارهای بی‌پایان (که تنها با دستوراتی مانند `break` متوقف می‌شود) به‌طور خاص مناسب ساخته است.

⁵⁴ Loops

ساختار کلی حلقه `while`:

While condition:

statement

نکته: دندانه‌گذاری در بدن حلقه (بخش دستورالعمل‌ها) ضروری است.

۶-۱) مثال‌های مربوط به حلقه **while**

مثال ۱. با استفاده از حلقه while برنامه بنویسید که اعداد ۱ تا ۵ چاپ شود.

python example - loops.py

```
1 # loops in Python (حلقه ها در پایتون)
2 # While loops in Python Programming
3
4 # Example 1:
5 count = 1
6 while count <= 5:
7     print(count)
8     count += 1
```

شكل ۶-۱- مثال چاپ اعداد ۱ تا ۵ با استفاده از حلقه **while**

خروجی شکل ۶-۱:

2

3

4

5

توضیحات شکل ۱-۶:

۱. تعریف متغیر: ابتدا متغیر `count` با مقدار ۱ تعریف می‌شود.
۲. شروع حلقه و برقرار بودن شرط حلقه: حلقه‌ی `while` برسی می‌کند که آیا مقدار `count` کمتر یا مساوی است. اگر شرط برقرار باشد، حلقه اجرا می‌شود.
۳. چاپ مقدار و افزایش متغیر: درون حلقه، ابتدا مقدار `count` چاپ می‌شود و سپس مقدار آن یک واحد افزایش می‌یابد.
۴. پایان حلقه: زمانی که مقدار `count` به ۶ برسد، شرط `count <= 5` برقرار نیست و حلقه خاتمه می‌یابد. این یک نمونه از حلقه‌ی تکراری است که در هر تکرار، مقدار متغیر را تغییر می‌دهد تا در نهایت به شرط خاتمه برسد و از حلقه خارج شود.
۵. با استفاده از حلقه while برنامه شمارش معکوس از ۱۰ به ۱ بنویسید.

python example - loops.py

```
12 # loops in Python (حلقه ها در پایتون)  
13 # While loops in Python Programming  
14  
15 # Example 2:  
16 countdown = 10  
17 while countdown > 0:  
18     print(countdown)  
19     countdown -= 1  
20 print("End of Counting!")  
21
```

شکل ۶-۲- مثال مربوط به شمارش معکوس با استفاده از حلقه *while*

خروجی شکل ۶-۲:

10

9

8

7

6

5

4

3

2

۱

End of Counting!

توضیحات شکل ۲-۶:

۱. تعریف متغیر: متغیر `countdown` با مقدار اولیه ۱۰ تعریف شده است.
۲. شروع حلقه‌ی `while` و برقراری شرط حلقه: حلقه‌ی `while` برسی می‌کند که آیا مقدار `countdown` بزرگتر از صفر است. اگر این شرط برقرار باشد، حلقه اجرا می‌شود.
۳. چاپ مقدار و کاهش متغیر: در هر تکرار، مقدار فعلی `countdown` چاپ می‌شود و سپس مقدار آن یک واحد کاهش می‌یابد.
۴. پایان حلقه: زمانی که مقدار `countdown` به صفر می‌رسد، شرط `< 0` برقرار نیست و حلقه خاتمه می‌یابد.
۵. پیام پایان: پس از اتمام حلقه، عبارت "پایان شمارش!" چاپ می‌شود.

این نمونه‌ای از یک شمارش معکوس است که در هر مرحله یک واحد از مقدار اولیه کم می‌کند و در پایان پیام مشخصی را نمایش می‌دهد.

مثال ۳. با استفاده از حلقه while برنامه محاسبه فاکتوریل یک عدد را بنویسید.

python example - loops.py

```
24 # loops in Python (حلقه ها در پایتون)
25 # While loops in Python Programming
26
27 # Example 3:
28 number = int(input("Enter a number: "))
29 factorial = 1
30 count = 1
31 while count <= number:
32     factorial *= count
33     count += 1
34 print(f"Factorial of {number} is: {factorial}")
```

شکل ۳-۶- مثال مربوط به محاسبه فاکتوریل یک عدد با استفاده از حلقه‌ی *while*

خروجی شکل ۳-۶:

Enter a number: 5

Factorial of 5 is: 120

توضیحات شکل ۳-۶:

۱. دریافت عدد: ابتدا عددی از کاربر دریافت می‌شود(با استفاده از تابع `input`) و به متغیر `number` اختصاص می‌یابد.

۲. تعریف متغیرها: متغیر `factorial` با مقدار اولیه ۱ و متغیر `count` با مقدار اولیه ۱ تعریف می‌شوند.

۳. حلقه `while` : حلقه‌ی `while` تا زمانی که مقدار `count` کمتر یا مساوی `number` باشد، تکرار می‌شود.

در هر تکرار، مقدار `count` در `factorial` ضرب شده و سپس `count` یک واحد افزایش می‌یابد.

۴. پایان حلقه و نمایش نتیجه: زمانی که مقدار `count` از `number` بیشتر شود، حلقه متوقف می‌شود و

فاکتوریل محاسبه شده در `factorial` نمایش داده می‌شود.

مثال ورودی و خروجی

برای مثال، اگر کاربر عدد ۵ را وارد کند:

۱. تکرار ۱ : `factorial = 1 * 1 = 1`

۲. تکرار ۲ : `factorial = 1 * 2 = 2`

۳. تکرار ۳ : `factorial = 2 * 3 = 6`

۴. تکرار ۴ : `factorial = 6 * 4 = 24`

۵. تکرار ۵ : `factorial = 24 * 5 = 120`

در نتیجه، خروجی به صورت زیر خواهد بود:

"۱۲۰" فاکتوریل ۵ برابر است با:

۲-۱-۶ بخش else در حلقه while

در زبان برنامه‌نویسی پایتون، از بخش `else` در ترکیب با حلقه‌ی `while` می‌توان برای اجرای یک بخش از کد

استفاده کرد که تنها زمانی اجرا می‌شود که حلقه به پایان طبیعی خود برسد؛ به عبارتی، اگر شرط حلقه‌ی `while`

در نهایت به "نادرست یا False" تبدیل شود و حلقه متوقف شود، بلاک کد در بخش `else` اجرا خواهد شد. این

ساختار در مواردی مفید است که بخواهیم مطمئن شویم عملیاتی پس از اتمام حلقه، در صورت برآورده شدن تمام شرایط، انجام می‌شود.

به طور دقیق‌تر، زمانی که یک حلقه‌ی `while` شروع به کار می‌کند، شرط آن بررسی می‌شود؛ اگر شرط برقرار باشد، کد درون حلقه اجرا می‌شود و این فرآیند تا زمانی ادامه می‌یابد که شرط نقض شود. پس از پایان حلقه، در صورتی که حلقه با استفاده از دستوری مانند `break` به اجبار متوقف نشده باشد، بخش `else` فعال شده و کد درون آن اجرا می‌شود. ساختار کلی حلقه `while` همراه با بخش `:else`:

While condition:

statement

else:

statement

برای مثال، فرض کنید می‌خواهید در یک حلقه، عددی را تا زمانی که کمتر از ۵ است افزایش دهید. اگر این عدد به ۵ یا بیشتر برسد، حلقه به‌طور طبیعی خاتمه می‌یابد و بخش `else` اجرا می‌شود. اما اگر به‌دلیل شرایط خاصی از حلقه خارج شوید (مانند استفاده از `break`)، بخش `else` نادیده گرفته خواهد شد. این ویژگی این امکان را می‌دهد تا بین خاتمه‌ی طبیعی و خاتمه‌ی غیرمنتظره‌ی حلقه تمایز قائل شوید.

به این ترتیب، "بخش `else` در حلقه‌ی `while`" ابزار ارزشمندی برای ساختاردهی و کنترل جریان اجرای کد است و می‌تواند برنامه‌نویسی را انعطاف‌پذیرتر و خواناتر سازد.

python example - loops.py

```
38 # loops in Python (حلقه ها در پایتون)
39 # While loops in Python Programming
40
41 # ** Else Statement in While loop **
42
43 count = 1
44 while count <= 5:
45     print(count)
46     count += 1
47 else:
48     print("End of the Loop!")
```

شکل ۶-۴- مثال مربوط به بخش *else* در حلقه *while*:

خروجی شکل ۶-۶:

1

2

3

4

5

End of the Loop!

۳-۱-۶) دستور **while** و **continue** و **break** در حلقه

در زبان برنامه‌نویسی پایتون، دستورهای `break` و `continue` به عنوان دو ابزار کلیدی برای کنترل جریان اجرای حلقه‌ها، به ویژه در ترکیب با حلقه‌ی `while`، مورد استفاده قرار می‌گیرند. این دستورات به برنامه‌نویسان امکان می‌دهند تا جریان طبیعی اجرای حلقه را بنا به شرایط مختلف تغییر دهند؛ به طوری که با استفاده از آن‌ها، می‌توان از ادامه یا خاتمه‌ی زودهنگام حلقه جلوگیری کرد و کد را به صورت بهینه‌تری کنترل نمود.

۳-۱-۶) دستور **break**

دستور `break` زمانی به کار می‌رود که حلقه، در لحظه‌ای خاص، حتی اگر شرط حلقه همچنان برقرار باشد، به اجراب خاتمه یابد. به عبارت دیگر، با نوشتن و اجرای این دستور، حلقه فوراً متوقف می‌شود و کنترل برنامه به خط بعد از حلقه منتقل می‌شود. از این دستور می‌توان در شرایطی استفاده کرد که اجرای حلقه براساس یک دستور خاص نیاز به توقف داشته باشد.

به عنوان مثال، فرض کنید در حلقه‌ای می‌خواهید عددی را بررسی کنید و به محض پیدا کردن یک مقدار خاص، از حلقه خارج شوید. در این حالت، دستور `break` به شما اجازه می‌دهد که با رسیدن به مقدار مورد نظر، حلقه را خاتمه دهید.

python example - loops.py

```
52 # loops in Python (حلقه ها در پایتون)
53 # While loops in Python Programming
54
55 # ** Break in While Loop **
56
57 number = 0
58 while True:
59     number += 1
60     print(f"Checking number: {number}")
61     if number == 5:
62         print("Number 5 found, exiting the loop.")
63         break
64 print("Loop ended.")
```

شکل ۶-۵- مثال مربوط به کلیدوازه *break* در حلقه‌ی *while*

خروجی شکل ۶-۵:

Checking number: 1

Checking number: 2

Checking number: 3

Checking number: 4

Checking number: 5

Number 5 found, exiting the loop.

Loop ended.

توضیحات شکل ۶-۵:

در این مثال، حلقه while به طور مداوم اجرا می‌شود(در خط اول شرط به اسن صورت است که تا زمانی که شرط درست باشد(حلقه بی‌نهایت)) و عدد را به طور افزایشی بررسی می‌کند. هنگامی که مقدار number به ۵ برسد، دستور break اجرا می‌شود و حلقه خاتمه می‌یابد.

۶-۳-۲) دستور `continue`

دستور `continue`، بر خلاف `break`، موجب توقف کامل حلقه نمی‌شود، بلکه تنها اجرای آن مرحله خاص از حلقه را متوقف می‌کند و کنترل را به ابتدای حلقه بازمی‌گرداند تا از مرحله بعدی آغاز شود. به بیان دیگر، این دستور تنها همان مرحله جاری حلقه را نادیده می‌گیرد و به مرحله بعد می‌رود. از این ویژگی برای صرفنظر کردن از مراحل غیرضروری یا خاص و تمرکز بر مراحل اصلی درون حلقه استفاده می‌شود.

برای مثال، اگر بخواهید در یک حلقه‌ی `while` تنها مقادیری را که مضربی از ۳ نیستند بررسی کنید، می‌توانید از `continue` استفاده کنید تا برنامه به محض برخورد با مضرب ۳، آن مرحله را رد کرده و به مرحله بعدی برود.

python example - loops.py

```
68 # loops in Python (حلقه ها در پایتون)
69 # While loops in Python Programming
70
71 # ** Continue in While Loop **
72
73 number = 0
74 while number < 10:
75     number += 1
76     if number % 3 == 0:
77         continue
78     print(f"Checking number: {number}")
```

شکل ۶-۶- مثال مربوط به کلیدواژه *continue* در حلقه‌ی *while*

خروجی شکل ۶-۶:

Checking number: 1

Checking number: 2

Checking number: 4

Checking number: 5

Checking number: 7

Checking number: 8

Checking number: 10

توضیحات شکل ۶-۶:

- مقدار `number` در هر مرحله افزایش می‌یابد.
 - در صورتی که `number` مضرب ۳ باشد، دستور `continue` اجرا می‌شود و مرحله جاری از حلقه نادیده گرفته شده و حلقه به مرحله بعدی می‌رود.
 - در نتیجه، فقط مقادیری چاپ می‌شوند که مضربی از ۳ نیستند.
- در نتیجه، دستورات `break` و `continue` از ابزارهای مهم در کنترل جریان حلقه‌های `while` در پایتون هستند که به کمک آن‌ها می‌توان انعطاف و کارآیی کد را افزایش داد و از تکرارهای غیرضروری جلوگیری کرد.

۲-۶) حلقه `for` در پایتون

در زبان برنامه‌نویسی پایتون، حلقه‌ی `for` ابزاری قدرتمند و انعطاف‌پذیر برای اجرای تکرارهای پی‌درپی در یک دنباله‌ی از پیش تعیین‌شده است. این دنباله می‌تواند شامل فهرست‌ها، مجموعه‌ها، رشته‌ها، یا حتی یک دامنه از اعداد باشد (نوع داده‌های که قابلیت پیمایش دارند). بر خلاف حلقه‌ی `while` که مبتنی بر یک شرط است، حلقه‌ی `for` به نحوی طراحی شده که بر روی عناصر یک دنباله حرکت کرده و هر عنصر را به ترتیب در متغیر خاصی ذخیره و اجرا کند. ساختار حلقه‌ی `for` در پایتون به این شکل است که با استفاده از عبارت `for item` خاصی ذخیره و اجرا کند. ساختار حلقه‌ی `for` در پایتون به این شکل است که با استفاده از عبارت `in sequence` تعریف می‌شود، به‌طوری‌که در هر بار اجرای حلقه، متغیر `item` مقدار بعدی در `sequence` را دریافت می‌کند. بدنه‌ی حلقه، که کدهای مربوط به هر تکرار را شامل می‌شود، در هر مرحله با مقدار جدید `item`: اجرا می‌شود تا زمانی که تمام عناصر درون دنباله مورد پردازش قرار گیرند. ساختار کلی حلقه `for`:

for item in sequence:

statement

نکته: دندانه‌گذاری در بدن حلقه (بخش دستورالعمل‌ها) ضروری است.

برای درک بهتر، فرض کنید می‌خواهیم اعداد ۱ تا ۵ را در یک فهرست چاپ کنیم. به‌کمک حلقه‌ی `for` می‌توانیم به‌سادگی به هر عدد دسترسی پیدا کرده و آن را نمایش دهیم. به عنوان مثال:

python example - loops.py

```
84 # loops in Python (حلقه‌ها در پایتون)  
85 # for loops in Python Programming  
86  
87 # Example 1:  
88 for number in [1,2,3,4,5]:  
89     print(number)
```

شكل ۶-۷- مثال مربوط به چاپ اعداد ۱ تا ۵ با استفاده از حلقه *for*

خروجی شکل ۶-۷:

-
- 1
 - 2
 - 3
 - 4
 - 5
-

توضیحات شکل ۷-۶:

در این مثال، در هر تکرار مقدار جدیدی از فهرست به متغیر `number` اختصاص می‌یابد و سپس چاپ می‌شود. این ویژگی حلقه‌ی `for` را برای پیمایش در مجموعه‌های داده و اعمال عملیات‌های مختلف بر روی آن‌ها، مانند محاسبه، نمایش یا اصلاح، به ابزاری ایده‌آل تبدیل می‌کند.

۱-۲-۶) تابع `range`

برای ایجاد یک دنباله‌ی عددی خاص و پیمایش آن در حلقه‌ی `for`، از تابع `range` استفاده می‌شود. این تابع به ما امکان می‌دهد تا یک دنباله‌ی پیوسته از اعداد با شروع، پایان و گام مشخص ایجاد کنیم. برای مثال، کد زیر از حلقه‌ی `for` به همراه `range` برای نمایش اعداد ۰ تا ۴ استفاده می‌کند:

python example - loops.py

```
93 # حلقه ها در پایتون()
94 # for loops in Python Programming
95
96 # Range in loops
97
98 for i in range(5):
99     print(f"This number is: {i}")
```

شکل ۶-۸- مثال مربوط به استفاده از تابع *range* در حلقه *for*

خروجی شکل ۶-۸:

This number is: 0

This number is: 1

This number is: 2

This number is: 3

This number is: 4

در نتیجه، حلقه‌ی `for` در پایتون با سادگی و کارآیی بالا برای کار با داده‌ها و تکرارهای پیوسته طراحی شده و از آن در برنامه‌های مختلف بهویژه در پردازش داده‌ها، جمع‌آوری اطلاعات و اعمال عملیات بر مجموعه‌های بزرگ استفاده می‌شود.

۲-۲-۶ بخش else در حلقه for

در زبان برنامه‌نویسی پایتون، بخش else در حلقه‌ی for ویژگی‌ای جذاب و گاهی نادیده گرفته شده است که امکان اجرای یک بخش کد پس از اتمام طبیعی حلقه را فراهم می‌سازد. این بخش تنها زمانی اجرا می‌شود که حلقه بدون استفاده از دستور break و به‌طور کامل به پایان رسیده باشد. اگر در حین اجرای حلقه، به دلیل وقوع شرط خاصی از break استفاده شود و حلقه پیش از پردازش تمام عناصر خاتمه یابد، بخش else نادیده گرفته می‌شود و اجرا نخواهد شد.

این ویژگی زمانی مفید است که بخواهیم بررسی کنیم آیا حلقه تمام عناصر دنباله یا مجموعه را به‌طور کامل پیموده یا خیر. به عنوان مثال، در جستجوی یک عنصر خاص در یک فهرست می‌توان از حلقه‌ی for برای بررسی تک‌تک عناصر استفاده کرد و در صورت یافتن عنصر، با break از حلقه خارج شد. اگر پس از بررسی تمام عناصر،

عنصر مورد نظر یافت نشد و حلقه به طور کامل اجرا گردید، بخش `else` به ما این امکان را می‌دهد که پیامی مبنی بر عدم وجود عنصر خاص در مجموعه به کاربر ارائه داد.

ساختار کلی حلقه `for` به همراه بخش `:else`:

for item in sequence:

statement

else:

statement

شکل ۹-۶، این مفهوم را به خوبی روشن می‌سازد:

python example - loops.py

```
103 # loops in Python (حلقه ها در پایتون)  
104 # for loops in Python Programming  
105  
106 # ** Else Statement in For loop **  
107  
108 items = [2, 4, 6, 8]  
109 target = 5  
110 for item in items:  
111     if item == target:  
112         print("Not Found the Number!:(")  
113         break  
114     else:  
115         print("list index out of range!:")
```

شکل ۹-۶- مثال مربوط به استفاده از بخش *else* در حلقه *for*

خروجی شکل ۹-۶:

list index out of range!:

توضیحات شکل ۹-۶:

در این مثال، حلقه تمام عناصر فهرست را بررسی می‌کند. اگر target یافت شود، حلقه با break به پایان می‌رسد و بخش else نادیده گرفته می‌شود. اما اگر پس از بررسی تمام عناصر، target موجود نباشد، حلقه به صورت طبیعی به پایان می‌رسد و بخش else اجرا خواهد شد.

۳-۲-۶) مثال‌های مربوط به حلقه‌ی for

مثال ۱. با استفاده از حلقه‌ی for میانگین نمرات دانشجو را محاسبه کنید.

برای محاسبه‌ی میانگین نمرات یک دانشجو با استفاده از حلقه‌ی `for`، می‌توانیم نمرات را در یک فهرست قرار دهیم، سپس با یک حلقه‌ی `for` تک‌تک نمرات را با هم جمع کرده و در نهایت میانگین آن‌ها را محاسبه کنیم. کد زیر یک نمونه از این روش را نشان می‌دهد:

python example - loops.py

```
119 # حلقة ها در پایتون)
120 # for loops in Python Programming
121
122 # Example 1: Student's Score
123
124 grades = [18, 20, 17, 19, 15] # لیست نمرات دانشجو
125
126 total = 0 # متغیر برای جمع نمرات
127
128 for grade in grades: # حلقة برای جمع کردن نمرات
129     total += grade
130
131 average = total / len(grades) # محاسبه میانگین
132
133 print("Avrage of Scores: ", average) # نمایش میانگین
```

شکل ۶-۱۰- مثال مربوط به محاسبه میانگین نمرات یک دانشجو با استفاده از حلقه for

Average of Scores: 17.8

توضیحات شکل ۶-۱۰:

ابتدا فهرستی به نام `grades` برای ذخیره‌ی نمرات دانشجو تعریف شده است. سپس یک متغیر به نام `total` برای جمع نمرات ایجاد کرده‌ایم و مقدار اولیه‌ی آن را برابر با صفر قرار داده‌ایم. با استفاده از حلقه‌ی `for`، تک‌تک نمرات را از فهرست `grades` دریافت کرده و به `total` اضافه می‌کنیم. در نهایت، برای محاسبه‌ی میانگین، جمع کل نمرات را بر تعداد نمرات تقسیم کرده و نتیجه را در `average` ذخیره می‌کنیم.

مثال ۲. با استفاده از حلقه `for`، اعداد زوج و فرد را از یک فهرست اولیه جدا کرده و هر کدام را داخل فهرست جداگانه قرار دهید.

python example - loops.py

```
137 # حلقه ها در پایتون (Loops in Python)
138 # for loops in Python Programming
139
140 # Example 2: Separate Even and Odd numbers from a list
141
142 main_numbers_list = [45, 2, 49, 17, 65, 98, 11, 83, 15, 32] # فهرست اعداد تصادفی
143
144 even_numbers_list = [] # ایجاد فهرست برای افزودن اعداد زوج
145 odd_numbers_list = [] # ایجاد فهرست برای افزودن اعداد فرد
146
147 for i in main_numbers_list: # استفاده از حلقه برای پیمایش بر روی فهرست اعداد تصادفی
148     if i%2 == 0: # استفاده از دستورات شرطی برای یافتن اعداد زوج و فرد و جداسازی آنها
149         even_numbers_list.append(i)
150     elif i%2 != 0:
151         odd_numbers_list.append(i)
152     else:
153         print("The number is not even or odd!")
154
155 # چاپ کردن فهرست اعداد زوج و فرد به صورت جداگانه
156 print(f"The Even numbers list is: {even_numbers_list}")
157 print(f"The Odd numbers list is: {odd_numbers_list}")
```

شکل ۶-۱۱- مثال مربوط به جداسازی اعداد زوج و فرد با استفاده از حلقه *for*

خروجی شکل ۶-۱۱:

The Even numbers list is: [2, 98, 32]

The Odd numbers list is: [45, 49, 17, 65, 11, 83, 15]

توضیحات شکل ۶-۱۱:

ابتدا یک فهرست از اعداد تصادفی ایجاد کرده و سپس دو فهرست برای اعداد زوج و فرد به صورت جداگانه ایجاد می‌کنیم. سپس یک حلقه for برای پیمایش بر روی فهرست اعداد تصادفی ایجاد کرده و در درون این حلقه، از دستور شرطی if برای یافتن اعداد زوج و فرد و جدا کردن آن‌ها استفاده می‌کنیم. در نهایت فهرست اعداد زوج و فرد را با استفاده از دستور print در خروجی چاپ می‌کنیم.

مثال ۳. با استفاده از حلقه for، کاراکترهای یک رشته را جدا کرده و چاپ کنید.

python example - loops.py

```
161 # loops in Python (حلقه ها در پایتون)
162 # for loops in Python Programming
163
164 # Example 3: Separate Character of String.
165
166 string_example = "Apples"
167 char_list = []
168
169 for char in string_example:
170     char_list.append(char)
171     print(f"The character in this loop is: {char}")
172
173 print(f"List of Characters: {char_list}")
```

شكل ۱۲-۶- مثال مربوط به جداسازی حروف یک رشته با استفاده از حلقه for

خروجی شکل ۱۲-۶ :

The character in this loop is: A

The character in this loop is: p

The character in this loop is: p

The character in this loop is: l

The character in this loop is: e

The character in this loop is: s

List of Characters: ['A', 'p', 'p', 'l', 'e', 's']

توضیحات شکل ۱۲-۶:

ابتدا یک رشته‌ی تصادفی مانند Apples string_example را ایجاد کرده و درون متغیر قرار می‌دهیم. همچنین یک فهرست برای افزودن کاراکترهای رشته ایجاد می‌کنیم. برای پیمایش بر روی رشته ایجاد شده باید از حلقه for استفاده کنیم. در هر تکرار، کاراکتر مورد نظر آن رشته ابتدا به فهرست ایجاد شده اضافه می‌شود و سپس آن کاراکتر در خروجی چاپ می‌شود. در نهایت، فهرستی که برای افزودن کاراکترهای رشته ایجاد کردیم را در خروجی چاپ می‌کنیم.

مثال ۴. با استفاده از حلقه for، کلید و مقدار یک دیکشنری را جدا کرده و در خروجی چاپ کنید.

python example - loops.py

```
177 # loops in Python (حلقه ها در پایتون)
178 # for loops in Python Programming
179
180 # Example 4: Separate keys and values from a Dictionary.
181 main_dict = {
182     "name": "Reza",
183     "age": 23,
184     "job": "Student",
185     "mobile": "09011111111",
186     "favorite color": "yellow"
187 }
188 keys_list = []
189 values_list = []
190
191 for key, value in main_dict.items():
192     keys_list.append(key)
193     values_list.append(value)
194
195 print(f"The Keys of main_dict are: {keys_list}")
196 print(f"The Values of main_dict are: {values_list}")
```

شكل ۱۳-۶- مثال مربوط به جداسازی کلید و مقدار یک دیکشنری با استفاده از حلقه *for*

خروجی شکل ۱۳-۶ :

The Keys of main_dict are: ['name', 'age', 'job', 'mobile', 'favorite color']

The Values of main_dict are: ['Reza', 23, 'Student', '09011111111', 'yellow']

توضیحات شکل ۱۳-۶ :

ابتدا یک دیکشنری تصادفی از کلید و مقدارهای تصادفی ایجاد می‌کنیم. سپس دو فهرست برای افزودن کلید و مقدار دیکشنری به صورت جداگانه ایجاد می‌کنیم. با استفاده از حلقه `for` و با استفاده از تابع `items`، به دیکشنری اختصاص می‌دهیم و در داخل حلقه، کلیدها و مقدارهای موجود در هر تکرار را به فهرست‌های ایجاد شده، اضافه می‌کنیم. در نهایت فهرست‌های ایجاد شده را در خروجی چاپ می‌کنیم.

مثال ۵. با استفاده از حلقه `for`، حاصل ضرب اعداد یک فهرست را محاسبه و نتیجه را در خروجی چاپ کنید.
برای محاسبه حاصل ضرب اعداد یک فهرست با استفاده از حلقه `'for'`، می‌توان از یک متغیر برای نگهداری حاصل ضرب استفاده کرد که مقدار اولیه آن برابر با ۱ باشد. سپس، در هر تکرار حلقه، مقدار عنصر جاری را در این متغیر ضرب کرد. در پایان حلقه، متغیر مورد نظر شامل حاصل ضرب همه اعداد خواهد بود. در ادامه، کد مربوطه را مشاهده می‌کنید:

python example - loops.py

```
200 # حلقه ها در پایتون (loops in Python)
201 # for loops in Python Programming
202
203 # Example 5: multiple of the numbers list
204
205 numbers_list = [2, 7, 1, 3] # فهرست نمونه‌ای از اعداد
206 multiple = 1 # متغیر برای ذخیره حاصل ضرب، مقدار اولیه ۱
207 for number in numbers_list:
208     print(f"This loop: {multiple} * {number}: {multiple*number}") # ضرب هر عنصر در متغیر multiple
209     multiple *= number
210
211 print("The multiple of the list is:", multiple)
```

شکل ۶-۱۴- مثال مربوط به محاسبه حاصل ضرب اعداد یک فهرست با استفاده از حلقه `for`

*This loop: 1 * 2: 2*

*This loop: 2 * 7: 14*

*This loop: 14 * 1: 14*

*This loop: 14 * 3: 42*

The multiple of the list is: 42

توضیحات شکل ۱۴-۶:

فهرستی از اعداد به نام `numbers` تعریف شده است. متغیر `multiple` با مقدار اولیه ۱ ایجاد شده است. در حلقه `for`، هر عنصر از `numbers` در `multiple` ضرب می‌شود. در نهایت، مقدار `multiple` که حاصل ضرب تمامی عناصر است، چاپ می‌شود.

۳-۶) حلقه‌های تو در تو

در زبان برنامه‌نویسی پایتون، حلقه‌های تو در تو^{۵۵} به حلقه‌هایی گفته می‌شود که درون حلقه‌ای دیگر تعریف می‌شوند. در این ساختار، اجرای هر دور از حلقه‌ی بیرونی منجر به اجرای کامل حلقه‌ی داخلی می‌شود؛ بدین معنی که حلقه‌ی داخلی به ازای هر بار اجرای حلقه‌ی بیرونی، به‌طور کامل پردازش خواهد شد. حلقه‌های تو در تو

^{۵۵} Nested Loops

به ویژه در مواقعی کاربرد دارند که نیاز به پردازش داده‌ها در سطوح چندگانه باشد؛ برای مثال، در کار با ماتریس‌ها، آرایه‌های چندبعدی، و یا ساختارهای پیچیده‌ای که شامل لایه‌های متعدد از داده هستند.

۶-۳-۱) شیوه‌ی کار حلقه‌های تو در تو

فرض کنید از دو حلقه‌ی for استفاده کردایم، به‌طوری‌که یکی به‌عنوان حلقه‌ی بیرونی و دیگری به‌عنوان حلقه‌ی داخلی عمل کند. در هر بار اجرای یک دور از حلقه‌ی بیرونی، حلقه‌ی داخلی به‌طور کامل اجرا خواهد شد. این روند تا زمانی ادامه می‌یابد که تمام دورهای حلقه‌ی بیرونی تکمیل شوند. به همین دلیل، تعداد کل تکرارها در حلقه‌های تو در تو معمولاً حاصل ضرب تعداد تکرارهای هر حلقه است.

مثال زیر نشان می‌دهد چگونه می‌توان از دو حلقه‌ی for تو در تو برای نمایش تمام ترکیب‌های ممکن از دو فهرست مجزا استفاده کرد:

python example - loops.py

```
215 # حلقه ها در پایتون)
216 # for loops in Python Programming
217
218 # ** Nested Loops **
219
220 for i in range(1, 4):
221     for j in range(1, 3):
222         print(f"i: {i}, j: {j}")
```

شکل ۶-۱۵- مثال حلقه‌های تو در تو

خروجی شکل ۶-۱۵:

i: 1, j: 1

i: 1, j: 2

i: 2, j: 1

i: 2, j: 2

i: 3, j: 1

i: 3, j: 2

توضیحات شکل ۱۵-۶:

در این مثال، حلقه‌ی بیرونی سه بار اجرا می‌شود(تکرارهای ۱ و ۲ و ۳)، و در هر بار از آن، حلقه‌ی داخلی دو بار تکرار خواهد شد(تکرارهای ۱ و ۲). بنابراین، این کد به‌طور کلی شش ترکیب مختلف از مقادیر i و j را نمایش خواهد داد.

۶-۳-۲) کاربردها و موارد استفاده حلقه‌های تو در تو

حلقه‌های تو در تو برای پردازش ساختارهای داده‌ی پیچیده، مانند ماتریس‌ها، جدول‌ها یا آرایه‌های چندبعدی بسیار مناسب هستند. برای نمونه، در تحلیل یک ماتریس دوبععدی (که دارای ردیف‌ها و ستون‌های متعدد است)، از حلقه‌های تو در تو برای دسترسی به هر عنصر ماتریس استفاده می‌شود. به علاوه، در بسیاری از الگوریتم‌های جستجو و مرتب‌سازی که نیازمند مقایسه‌های چندگانه میان عناصر هستند، حلقه‌های تو در تو از ابزارهای اساسی محسوب می‌شوند.

فصل هفتم

توابع در زبان برنامه‌نویسی پایتون

۷) مقدمه‌ای بر توابع در زبان برنامه‌نویسی پایتون

در زبان برنامه‌نویسی پایتون، "تتابع^{۵۶}" به عنوان واحدهای مستقلی از کد تعریف می‌شوند که وظایف مشخصی را برعهده دارند و به کد اجازه می‌دهند تا ساختاری منظم، خوانا و قابل استفاده مجدد داشته باشد. تتابع ابزاری کارآمد برای تجزیه‌ی برنامه‌های پیچیده به بخش‌های کوچک‌تر و مدیریت آسان‌تر آن‌ها فراهم می‌کنند و امکان استفاده مجدد از کد و کاهش خطاهای احتمالی را نیز فراهم می‌سازند.

۷-۱) تعریف و هدف از تتابع

در حقیقت، تابع مجموعه‌ای از دستورات است که تحت نام خاص تعریف می‌شود و برای اجرای یک وظیفه‌ی خاص به کار می‌رود. تتابع به برنامه‌نویس این امکان را می‌دهند که با یکبار نوشتن مجموعه‌ای از دستورات، از آن‌ها در بخش‌های مختلف برنامه استفاده کنند. هر تابع می‌تواند مقادیری را به عنوان ورودی بپذیرد و پس از پردازش این داده‌ها، نتیجه‌ای را بازگرداند. در پایتون، تعریف تابع با استفاده از کلمه‌ی کلیدی `def` صورت می‌گیرد و پس از آن، نام تابع و پارامترهای ورودی (در صورت نیاز) در داخل پرانتزها قرار می‌گیرند.

برای مثال، تابع ساده‌ای که یک عدد را دو برابر می‌کند به صورت زیر تعریف می‌شود:

```
def double(number):
```

```
    return number * 2
```

در این تابع، نام تابع را به دلخواه `double` گذاشته و مقدار ورودی `number` دریافت کرده و سپس دو برابر شده و نتیجه با استفاده از دستور `return` به برنامه بازگردانده می‌شود.

^{۵۶} functions

نکته: معمولاً دستور `return`، در خروجی چیزی را چاپ نمی‌کند و صرفاً آن را در حافظه نگهداری می‌کند، برای چاپ در خروجی (ترمینال) باید نام تابع را در داخل تابع داخلی `print` استفاده کرد (نحوه اجرا کردن توابع در قسمت‌های بعدی اشاره خواهد شد) و یا می‌توان به جای دستور `return` از دستور `print` استفاده کنید.

python example - functions.py

```
1 # Functions in Python (توابع در پایتون)
2
3 # Example:
4 def greet():
5     print("Hello, World!")
6
7 greet()
```

شکل ۷-۱- مثال چاپ *Hello, World!* با استفاده از تعریف تابع در پایتون

خروجی شکل ۷-۱:

Hello, World!

python example - functions.py

```
11 # Functions in Python (توابع در پایتون)
12 # Example:
13 def Sum(Score_List):
14     total = 0
15     for i in Score_List:
16         total += i
17     return total
18
19
20 myScores = [18, 17, 20, 11, 16]
21 print(f"Sum of myScores is {Sum(myScores)}")
```

شکل ۷-۲- مثال محاسبه مجموع اعداد داخل یک فهرست با استفاده از تعریف تابع

خروجی شکل ۷-۲:

Sum of myScores is 82

python example - functions.py

```
25 # # Functions in Python (توابع در پایتون)
26 # # Example:
27
28 def factorial(n):
29     if n == 0:
30         return 1
31     else:
32         return n * factorial(n-1)
33
34 number = int(input("please Enter integer number: "))
35 if number < 0:
36     print("Sorry, factorial does not exist for negative numbers")
37 else:
38     result = factorial(number)
39     print("The factorial of {} is: {}".format(number, result))
```

شکل ۷-۳- مثال محاسبه فاکتوریل یک عدد با استفاده از تعریف تابع

خروجی شکل ۷-۳:

please Enter integer number: 12

The factorial of 12 is: 479001600

۷-۲) مزایای استفاده از توابع

استفاده از توابع در برنامه‌نویسی چندین مزیت کلیدی دارد:

۱. توابع با جداسازی وظایف مختلف، کد را سازماندهی کرده و خوانایی آن را بهبود می‌بخشد.
۲. توابع به برنامه‌نویسان امکان می‌دهد که کد را یک بار نوشه و آن را در قسمت‌های مختلف برنامه استفاده کند.
۳. هنگامی که کد به بخش‌های کوچک‌تر تقسیم می‌شود، شناسایی و رفع خطای نیز آسان‌تر می‌شود.

۷-۳) توابع داخلی و توابع تعریف شده توسط کاربر

در پایتون، دو نوع کلی از توابع وجود دارد: "تتابع داخلی" و "تتابع تعریف شده توسط کاربر".

تتابع داخلی پایتون توابعی هستند که به طور پیش فرض در زبان برنامه نویسی موجودند و برای انجام کارهای رایج مانند محاسبات ریاضی، کار با رشته ها و پردازش فهرست ها استفاده می شوند؛ برای مثال `len()` برای محاسبه طول فهرست یا رشته و `sum()` برای جمع کردن عناصر یک فهرست. اما برنامه نویس می تواند تتابع خود را نیز ایجاد کند تا وظایف خاصی را انجام دهد که در برنامه نویسی و حل مسائل پیچیده بسیار کاربرد دارند.

به طور خلاصه، "تتابع" در پایتون ابزاری حیاتی و ضروری هستند که به برنامه نویسان امکان می دهند تا کدی تمیز، مؤثر و مقیاس پذیر بنویسند. تتابع با جداسازی بخش های مختلف کد و افزایش امکان استفاده مجدد از آن، برنامه نویسی را تسهیل می کنند و به دستیابی به ساختارهای قابل مدیریت تر کمک می کنند.

۷-۴) نحوه اجرا و فراخوانی تتابع

زمانی که تابعی با نام دلخواه ساختید، می توانید در خط جدید با فراخوانی نام تابع، از آن تابع استفاده کنید.

نکته: معمولاً زمانی که نام تابع همراه با پرانتز قرار می دهید، آن تابع را فرا می خوانید (در ادامه توضیحات بیشتری داده خواهد شد).

در زبان برنامه نویسی پایتون، "فراخوانی تابع" به دو شیوه متفاوت انجام می شود: "با پرانتز" و "بدون پرانتز". در ک تفاوت میان این دو شیوه ضروری است، زیرا هر یک نتایج متفاوتی به همراه دارند و در موقعیت های خاص خود به کار می روند.

۷-۴-۱) فراخوانی تابع با پرانتز

هنگامی که تابع را "با پرانتز" فراخوانی می‌کنید طبق شکل ۷-۴، در واقع به پایتون دستور می‌دهید که آن تابع را اجرا کند و نتیجه‌ی آن را بازگرداند. در این حالت، تابع به همراه پارامترهای ورودی (در صورت وجود) درون پرانتز قرار می‌گیرد و پایتون ابتدا تابع را اجرا و سپس نتیجه‌ی حاصل از اجرای آن را بازمی‌گرداند.

python example - functions.py

```
43 # Functions in Python)  
44  
45 # Function with () and without ()  
46 # With ()  
47  
48 def greet():  
49     print("Hello, World!")  
50  
51 greet()
```

شکل ۷-۴-۱) مثال مربوط به فراخوانی تابع با پرانتز در پایتون

خروجی شکل ۷-۴:

Hello, World!

توضیحات شکل ۷-۴:

در اینجا، `greet()` با پرانتز فراخوانی شده است، بنابراین تابع اجرا می‌شود و عبارت ``Hello, World`` به عنوان خروجی چاپ می‌شود.

۲-۴-۷) فراخوانی تابع بدون پرانتز

فراخوانی تابع "بدون پرانتز" به این معناست که به خود تابع (به عنوان یک شیء) اشاره می‌کنید و از اجرای آن صرف نظر می‌کنید. در این حالت، پایتون تابع را اجرا نمی‌کند؛ بلکه تنها یک ارجاع به تابع یا همان نام تابع را بازمی‌گرداند. این رویکرد زمانی مفید است که قصد دارید تابع را به عنوان یک آرگومان به تابع دیگری ارسال کنید یا زمانی که می‌خواهید تابع را بدون اجرای فوری، به متغیری اختصاص دهید مانند شکل ۵-۷.

python example - functions.py

```
55 # Functions in Python (توابع در پایتون)
56
57 # Function with () and without ()
58 # Without ()
59
60 def greet():
61     return "Hello, World!"
62
63 greeting = greet
64
65 print(greeting())
```

شکل ۷-۵- مثال مربوط به فراخوانی تابع بدون پرانتز در پایتون

خروجی شکل ۷-۵:

Hello, World!

توضیحات شکل ۷-۵:

در این مثال، وقتی `greeting = greet` نوشته می‌شود، به تابع `greet` بدون اجرای آن اشاره می‌کنید. سپس با استفاده از `greeting()` می‌توانید تابع را فراخوانی کرده و خروجی را دریافت کنید. اگر در همان ابتدا `greeting()` نوشته می‌شد، تابع `greet` بلافصله اجرا می‌شد و نتیجه‌ی آن در `greeting` ذخیره می‌شد.

یک متغیر، یک تابع بدون پرانتز داده شود، آن متغیر تبدیل به تابع می‌شود.

استفاده از پرانتز یا عدم استفاده از آن در فراخوانی توابع در پایتون نقش مهمی در تعیین رفتار برنامه دارد. "فراخوانی با پرانتز" تابع را اجرا و نتیجه را بازمی‌گرداند، در حالی که "فراخوانی بدون پرانتز" فقط به تابع اشاره می‌کند، بدون آنکه آن را اجرا کند. این تفاوت در کدنویسی پیشرفته و هنگام ارسال توابع به عنوان آرگومان یا اختصاص آن‌ها به متغیرها اهمیت ویژه‌ای دارد و انعطاف بالاتری در کد فراهم می‌سازد.

۷-۵) پارامترها و آرگومان‌ها در توابع

"پارامترها"^{۵۷} به عنوان ورودی‌های تابع عمل می‌کنند و به آن امکان می‌دهند تا با داده‌های مختلف کار کرده و رفتار خود را براساس نیاز تغییر دهد. در واقع، پارامترها متغیرهایی هستند که در هنگام تعریف تابع تعیین می‌شوند و مقادیری که تابع در زمان فراخوانی دریافت می‌کند، "آرگومان"^{۵۸} نامیده می‌شوند. این ویژگی توابع را انعطاف‌پذیر و قابل استفاده مجدد می‌کند، زیرا با استفاده از پارامترها، توابع می‌توانند با مقادیر مختلفی به کار گرفته شوند و نتایج متفاوتی را تولید کنند.

57 Parameters

⁵⁸ Arguments

۷-۵-۱) تعریف پارامترها

در زبان برنامه‌نویسی پایتون، پارامترها در داخل پرانتز پس از نام تابع نوشته می‌شوند و می‌توانند در طول اجرای تابع مورد استفاده قرار گیرند. برای مثال، طبق شکل ۷-۶، تابع یک پارامتر به نام `name` می‌پذیرد و با توجه به مقدار دریافتی، پیامی شخصی‌سازی‌شده ایجاد می‌کند.

python example - functions.py

```
69 # Functions in Python (توابع در پایتون)
70
71 # Argument and Parameter
72 # Parameter
73
74 def greet(name):
75     return f"Hello, {name}!"
76
77 print(greet("Alice"))
78 print(greet("Bob"))
```

شکل ۷-۶- مثالی از تعریف پارامتر برای تابع در پایتون

خروجی شکل ۷-۶:

Hello, Alice!

Hello, Bob

توضیحات شکل ۷-۶:

در اینجا، `name` پارامتری است که تابع `greet` دریافت می‌کند و هر بار که تابع فراخوانی می‌شود، مقدار متفاوتی به آن اختصاص داده می‌شود. این انعطاف‌پذیری، کد را خواناتر و مقیاس‌پذیرتر می‌سازد.

۱-۵-۷) انواع پارامترها در پایتون

پایتون امکان تعریف انواع مختلف پارامترها را فراهم می‌کند:

۱. پارامترهای موقعیتی^{۵۹}: این پارامترها بر اساس موقعیتی که به تابع ارسال می‌شوند مقداردهی می‌شوند. به عنوان مثال، در شکل ۷-۷، در تابع `add`، پارامترهای `x` و `y` به ترتیب با مقادیر `۱۰` و `۲۰` مقداردهی می‌شوند.

python example - functions.py

```
82 # Functions in Python (توابع در پایتون)
83 # Types of Parameters
84 # Positional Parameters پارامترهای موقعیتی
85
86 def add(x, y):
87     return x + y
88
89 print(add(10, 20))
```

شکل ۷-۷-مثالی از تعریف پارامتر موقعیتی در توابع پایتون

خروجی شکل ۷-۷:

30

⁵⁹ Positional Parameters

۲. پارامترهای کلیدواژه‌ای^{۶۰}: در این نوع، مقداردهی به پارامترها بر اساس نام آن‌ها انجام می‌شود. این رویکرد به خوانایی کد کمک می‌کند، مخصوصاً زمانی که تعداد پارامترها زیاد باشد. برای درک بهتر به شکل ۸-۷ توجه نمایید.

python example - functions.py

```
93 # Functions in Python)  
94  
95 # Types of Parameters  
96 # Keyword Parameters  
97  
98 def introduce(name, age):  
99     return f"My name is {name} and I am {age} years old."  
100  
101 print(introduce(age=25, name="Alice"))
```

شکل ۸-۷- مثالی از تعریف پارامتر کلیدواژه‌ای در توابع پایتون

خروجی شکل ۸-۷:

My name is Alice and I am 25 years old.

۳. پارامترهای پیش‌فرض^{۶۱}: این پارامترها دارای مقدار پیش‌فرضی هستند که اگر هنگام فراخوانی تابع مقداری به آن‌ها اختصاص داده نشود، از مقدار پیش‌فرض استفاده می‌شود. به شکل ۹-۷ توجه نمایید.

⁶⁰ Keyword Parameters

⁶¹ Default Parameters

python example - functions.py

```
105 # Functions in Python (توابع در پایتون)
106
107 # Types of Parameters
108 # Default Parameters پارامترهای پیش فرض
109
110 def greet(name="Guest"):
111     return f"Hello, {name}!"
112
113 print(greet())
114 print(greet("Alice"))
```

شکل ۷-۹- مثالی از تعریف پارامتر پیشفرض در توابع پایتون

خروجی شکل ۷-۹:

Hello, Guest!

Hello, Alice!

۴. پارامترهای دلخواه^{۶۲}: زمانی که تعداد پارامترها نامشخص باشد، می‌توان مانند شکل ۱۰-۷ از پارامترهای دلخواه استفاده کرد. با افزودن `*` قبل از نام پارامتر، این نوع پارامتر به صورت یک فهرست قابل دسترسی خواهد بود.

⁶² Arbitrary Parameters

python example - functions.py

```
118 # Functions in Python (توابع در پایتون)
119
120 # Types of Parameters
121 # Arbitrary Parameters پارامترهای دلخواه
122
123 def summarize(*args):
124     return sum(args)
125
126 print(f"Answer is: {summarize(1, 2, 3, 4)}")
```

شکل ۷-۱۰- مثالی از تعریف پارامتر دلخواه در توابع پایتون

خروجی شکل ۷-۱۰:

Answer is: 10

پارامترها در پایتون یکی از مهم‌ترین عناصر توابع محسوب می‌شوند که با ارائه‌ی انعطاف بالا و امکان سفارشی‌سازی رفتار توابع، قدرت و کارایی برنامه‌نویسی را افزایش می‌دهند. به کمک انواع مختلف پارامترها، برنامه‌نویسان می‌توانند توابع خود را برای طیف وسیعی از ورودی‌ها آماده و کدی مؤثر و قابل استفاده مجدد بنویسند.

همان‌طور که اشاره شد، "پارامترها" متغیرهایی هستند که هنگام تعریف تابع مشخص می‌شوند و نقشی مانند "ورودی‌های تابع" را دارند. حال، "آرگومان‌ها" به مقادیری اطلاق می‌شود که در زمان "فراخوانی تابع" به این

پارامترها اختصاص می‌یابد. به عبارت دیگر، پارامترها "ظرفهایی" هستند که در تعریفتابع تعیین می‌شوند، و آرگومان‌ها همان "محتویاتی" هستند که هنگام فراخوانی تابع، درون این ظرفها قرار می‌گیرند.

۲-۵-۷) تعریف آرگومان

۲-۵-۷) انواع آرگومان‌ها

پایتون انواع مختلفی از آرگومان‌ها را پشتیبانی می‌کند که به برنامه‌نویس امکان می‌دهند به صورت انعطاف‌پذیر با توابع کار کنند. این انواع شامل موارد زیر است:

۱. آرگومان‌های موقعیتی^{۶۳}: این آرگومان‌ها بر اساس موقعیت خود در زمان فراخوانی به پارامترهای تابع اختصاص داده می‌شوند. برای مثال:

python example - functions.py

```
130 # Functions in Python (توابع در پایتون)
131
132 # Types of Arguments
133 # Positional Arguments آرگومان‌های موقعیتی
134
135 def calculate_area(length, width):
136     return length * width # محاسبه مساحت یک مستطیل
137
138 area = calculate_area(5, 3) # طول = 5, عرض = 3
139 print(area)
```

شكل ۷-۱۱- مثالی از تعریف آرگومان موقعیتی در توابع پایتون

⁶³ Positional Arguments

15

۲. آرگومان‌های کلیدوازه‌ای^{۶۴}: این آرگومان‌ها بر اساس نام پارامترها مقداردهی می‌شوند، که خوانایی کد را بهبود می‌بخشد و این امکان را می‌دهند که ترتیب آرگومان‌ها را تغییر دهید:

python example - functions.py

```
144 # Functions in Python) (توابع در پایتون)
145
146 # Types of Arguments
147 # Keyword Arguments آرگومان کلیدوازه
148
149 def calculate_area(length, width):
150     return length * width # محاسبه مساحت یک مستطیل
151
152 area = calculate_area(length=5, width=3) # نام آرگومان‌ها مشخص شده
153 print(area)
```

شكل ۷-۱۲-۱۲-مثالی از تعریف آرگومان کلیدوازه‌ای در توابع پایتون

15

⁶⁴ Keyword Arguments

۳. آرگومان‌های دلخواه^{۶۵}: گاهی تعداد آرگومان‌های ورودی معلوم نیست؛ در این صورت می‌توان با استفاده از `*args` یا `**kwargs` این نوع آرگومان‌ها را به تابع ارسال کرد.

فهرستی از آرگومان‌های موقعیتی و `**kwargs` دیکشنری‌ای از آرگومان‌های کلیدواژه‌ای را دریافت می‌کند:

python example - functions.py

```
158 # Functions in Python (تابع در پایتون)
159
160 # Types of Arguments
161 # Arbitrary Arguments آرگومان دلخواه
162
163 def print_info(*args, **kwargs):
164     print("Positional Arguments:", args)
165     print("Keyword Arguments:", kwargs)
166
167 print_info("Alice", 30, city="New York", profession="Engineer")
```

شکل ۷-۱۳- مثالی از تعریف آرگومان دلخواه در تابع پایتون

خروجی شکل ۷-۱۳:

Positional Arguments: ('Alice', 30)

Keyword Arguments: {'city': 'New York', 'profession': 'Engineer'}

⁶⁵ Arbitrary Arguments

"پارامترها" به متغیرهایی اشاره دارند که هنگام تعریف تابع تعیین می‌شوند، در حالی که "آرگومان‌ها" مقادیری هستند که هنگام فراخوانی تابع به این پارامترها اختصاص می‌یابند. این تفاوت میان پارامتر و آرگومان به توابع پایتون قدرت بالایی می‌بخشد و به برنامه‌نویسان امکان می‌دهد توابعی چندمنظوره، انعطاف‌پذیر و قابل استفاده مجدد بسازند. برای درک بهتر این موضوع، به شکل

۳-۵-۷) مثال و تفاوت میان پارامترها و آرگومان‌ها

با توجه به شکل ۱۴-۷، فرض کنید تابعی به نام `introduce` دارد که دو پارامتر `name` و `age` را می‌پذیرد. هنگام تعریف تابع، `name` و `age` بعنوان پارامترها شناخته می‌شوند. اما در زمان فراخوانی تابع، مقادیر خاصی بعنوان آرگومان به این پارامترها ارسال می‌شوند.

python example - functions.py

```
171 # Functions in Python (توابع در پایتون)
172
173 # difference between argument and parameter in python
174
175 def introduce(name, age):
176     # name and age are Parameter
177     return f"My name is {name} and I am {age} years old."
178
179 print(introduce("Alice", 30))
180 # "Alice" and 30 are Argument
181
182 print(introduce("Reza", 24))
183 # "Reza" and 24 are Argument
```

شکل ۱۴-۷- مثالی از تفاوت بین پارامتر و آرگومان در توابع پایتون

: ۱۴-۷ خروجی شکل

My name is Alice and I am 30 years old.

My name is Reza and I am 24 years old

توضیحات شکل ۱۴-۷:

در این مثال، "Alice" و "۳۰" آرگومان‌هایی هستند که به ترتیب در پارامترهای `name` و `age` جایگذاری می‌شوند. با هر فراخوانی تابع، مقادیر آرگومان‌ها می‌توانند تغییر کنند و رفتار تابع بر اساس این ورودی‌های جدید، نتایج متفاوتی تولید می‌کند.

۴-۵-۷) آرگومان‌های دلخواه

"آرگومان‌های دلخواه" در پایتون به برنامه‌نویس این امکان را می‌دهند تا توابعی ایجاد کند که تعداد نامشخصی از ورودی‌ها را بپذیرند. این ویژگی زمانی کاربرد دارد که تابع باید بتواند ورودی‌های مختلفی را بدون محدودیت تعداد دریافت کند. پایتون با استفاده از دو نوع نشانه‌گذاری، این قابلیت را فراهم می‌سازد: `*args` برای آرگومان‌های موقعیتی دلخواه و `**kwargs` برای آرگومان‌های کلیدوازه‌ای دلخواه.

۴-۵-۸) آرگومان‌های دلخواه با `*args`

هنگامی که در تعریف تابع از `*args` استفاده می‌کنیم، پایتون هر تعداد آرگومان موقعیتی که در فراخوانی تابع قرار گیرد را به صورت یک "تاپل" در `args` جمع‌آوری می‌کند. با این روش می‌توان تعداد متغیرهای نامشخصی را به تابع ارسال کرد و تابع آن‌ها را به صورت فهرستی از ورودی‌ها دریافت می‌کند.

شکل ۱۵-۷، تابعی را نشان می‌دهد که چندین عدد را به عنوان ورودی دریافت می‌کند و جمع آن‌ها را محاسبه و

بازمی‌گرداند:

python example - functions.py

```
187 # Functions in Python (توابع در پایتون)  
188  
189 # Types of Arguments  
190 # Arbitrary Arguments (*args explain) آرگومان های دلخواه  
191 def sum_numbers(*args):  
192     total = sum(args)  
193     return total  
194  
195 print(sum_numbers(1, 2, 3, 4))  
196 print(sum_numbers(5, 10, 15))
```

شکل ۱۵-۷-۱۵- مثالی از آرگومان دلخواه **args*

خروجی شکل ۱۵-۷:

10

30

۱۵-۴-۲) آرگومان‌های کلیدوازه‌ای دلخواه با `**kwargs`

در حالتی که بخواهیم تعداد نامشخصی از "آرگومان‌های کلیدوازه‌ای" (به صورت `key=value`) را به تابع ارسال

کنیم، می‌توانیم از `**kwargs` استفاده کنیم. با این روش، تمامی آرگومان‌های کلیدوازه‌ای به صورت یک

"دیکشنری" در `kwargs` ذخیره می‌شوند. این امکان، کد را بسیار انعطاف‌پذیر می‌کند و به برنامه‌نویس اجازه می‌دهد تا تابعی داشته باشد که بتواند تعداد متغیرهای کلیدواژه‌ای را دریافت کند. به شکل ۱۶-۷ توجه کنید:

python example - functions.py

```
200 # Functions in Python (توابع در پایتون)  
201  
202 # Types of Arguments  
203 # Arbitrary Arguments (**kwargs explain)  
204  
205 def print_info(**kwargs):  
206     for key, value in kwargs.items():  
207         print(f"{key}: {value}")  
208  
209 print_info(name="Alice", age=30, city="New York")
```

شكل ۱۶-۷- مثالی از آرگومان دلخواه **kwargs

خروجی شکل ۱۶-۷:

name: Alice

age: 30

city: New York

توضیحات شکل ۱۶-۷:

در اینجا، `kwargs` دیکشنری‌ای شامل کلیدها و مقادیر مربوط به هر آرگومان کلیدوازه‌ای ورودی است. در فراخوانی تابع، اطلاعاتی چون `name`، `age` و `city` به `print_info` ارسال می‌شوند، و تابع با استفاده از یک حلقه آن‌ها را چاپ می‌کند.

۳-۴-۵-۷) ترکیب `*args` و `**kwargs` در یک تابع

گاهی اوقات ممکن است نیاز باشد تابعی داشته باشیم که هم آرگومان‌های موقعیتی و هم آرگومان‌های کلیدوازه‌ای دلخواه را بپذیرد. در این صورت، `*args` و `**kwargs` می‌توانند با هم ترکیب شوند؛ به شرط آنکه `*args` قبل از `**kwargs` در تعریف تابع قرار گیرد. شکل ۷-۷، ترکیب این دو نوع آرگومان را نشان می‌دهد:

python example - functions.py

```
213 # Functions in Python (توابع در پایتون)
214
215 # Types of Arguments
216 # Arbitrary Arguments (*args and **kwargs)
217
218 def display_info(*args, **kwargs):
219     print("Positional arguments:", args)
220     print("Keyword arguments:", kwargs)
221
222 display_info(1, 2, 3, name="Alice", age=30)
```

شکل ۷-۷-مثالی از ترکیب `*args` و `**kwargs`

خروجی شکل ۷-۷:

Positional arguments: (1, 2, 3)

Keyword arguments: `{'name': 'Alice', 'age': 30}`

به صورت خلاصه، "آرگومان‌های دلخواه" با استفاده از `*args` و `**kwargs`، به برنامه‌نویس این امکان را می‌دهند تا توابعی منعطف و پویا تعریف کند که قادر به پذیرش تعداد نامحدودی از ورودی‌ها باشند. این قابلیت برای ساختاردهی بهتر کد، و سازگاری توابع با ورودی‌های مختلف بسیار مفید است.

مثال ۱. جمع چندین عدد با استفاده `*args`

python example - functions.py

```
226 # Functions in Python) (توابع در پایتون)
227
228 # *args Examples
229 # Example 1: Sum many numbers using *args
230
231 def sum_all(*args):
232     total = 0
233     for number in args:
234         total += number
235     return total
236
237 print(f"Sum of the (1, 2, 3): {sum_all(1, 2, 3)}")
238 print(f"Sum of the (10, 20, 30, 40): {sum_all(10, 20, 30, 40)}")
```

شكل ۷-۱۸- مثالی از محاسبه چندین عدد با استفاده از `*args`

خروجی شکل ۷-۱۸:

Sum of the (1, 2, 3): 6

Sum of the (10, 20, 30, 40): 100

مثال ۲. نمایش یک پیغام با چندین نام با استفاده از `*args`

python example - functions.py

```
242 # Functions in Python (توابع در پایتون)
243
244 # *args Examples
245 # Example 2: Display a message and then several names using *args
246
247 def greet(message, *names):
248     for name in names:
249         print(f"{message}, {name}!")
250
251 greet("Hello", "Ali", "Sara", "Reza")
```

شكل ۷-۱۹- مثالی از نمایش یک پیام با اسمی مختلف با استفاده از `*args`

خروجی شکل ۷-۱۹:

Hello, Ali!

Hello, Sara!

Hello, Reza!

مثال ۳. نمایش اطلاعات یک کاربر با `**kwargs`

python example - functions.py

```
255 # Functions in Python (توابع در پایتون)  
256  
257 # **kwargs Examples  
258 # Example 1: Display a user's information using **kwargs  
259  
260 def display_user_info(**kwargs):  
261     for key, value in kwargs.items():  
262         print(f"{key}: {value}")  
263  
264 display_user_info(name="Ali", age=25, job="Engineer")
```

شکل ۷-۲۰- مثالی از نمایش اطلاعات یک کاربر با استفاده از `**kwargs`

خروجی شکل ۷-۲۰:

name: Ali

age: 25

job: Engineer

مثال ۴. ساخت یک پیام شخصی‌سازی شده با `**kwargs`.

```
268 # Functions in Python (توابع در پایتون)  
269  
270 # **kwargs Examples  
271 # Example 2: Create a personalized message using  
272  
273 def custom_message(template, **kwargs):  
274     return template.format(**kwargs)  
275  
276 message = custom_message("Hello {name}, you have {notifications} new notifications.",  
277                         name="Sara", notifications=5)  
278 print(message)
```

شکل ۷-۲۱- مثالی از ساخت یک پیام شخصی‌سازی‌شده با `**kwargs`

خروجی شکل ۷-۲۱:

Hello Sara, you have 5 new notifications

۶-۷) توابع درونی در زبان برنامه‌نویسی پایتون

"توابع درونی"^{۶۶} در زبان برنامه‌نویسی پایتون، مجموعه‌ای از توابع آماده و از پیش تعریف شده هستند که برای انجام عملیات پایه و ضروری طراحی شده‌اند و بدون نیاز به وارد کردن کتابخانه‌ای خاص، در دسترس برنامه‌نویسان قرار می‌گیرند. این تابع با هدف افزایش کارایی و ساده‌سازی کدنویسی پایتون تعییه شده‌اند و می‌توانند عملیات متنوعی چون پردازش‌های ریاضی، تبدیل داده، مدیریت فایل و حتی مدیریت ارور را به سهولت به انجام رسانند.

مثال‌های مهمی از این تابع شامل `print()` برای نمایش اطلاعات در خروجی، `len()` برای محاسبه طول یک دنباله، و `int()`, `float()`, `str()`, `type()` برای دریافت نوع داده‌ی متغیرها هستند. همچنین، توابعی همچون

^{۶۶} Built-in Functions

شما اجازه می‌دهند نوع داده‌ها را به سادگی به اعداد صحیح، اعداد اعشاری و یا رشته تبدیل کنید. دیگر توابع مانند `sum()` برای محاسبه‌ی مجموع عناصر یک دنباله، و یا `min()` و `max()` برای یافتن کوچکترین و بزرگ‌ترین مقدار در یک دنباله، نمونه‌هایی از این قابلیت‌های توابع درونی هستند.

توابع درونی پایتون در بسیاری از موارد برای حل مسائل روزمره برنامه‌نویسی طراحی شده‌اند. برای مثال، تابع `abs()` مقدار مطلق یک عدد را بازمی‌گرداند، و تابع `round()` امکان گرد کردن اعداد اعشاری را فراهم می‌کند. به کمک این توابع، برنامه‌نویس نیازی به تعریف توابع پیچیده و تکراری برای عملیات‌های رایج ندارد، که این موضوع بهینه‌سازی و خوانایی کد را افزایش می‌دهد.

این توابع، به سبب بهینه‌سازی و سرعت اجرایی بالا، برای برنامه‌هایی با حجم پردازش بالا نیز ایده‌آل‌اند و از طرفی با ترکیب آن‌ها با توابع دیگر یا در ساختارهای مختلف پایتون مانند فهرست‌ها، دیکشنری‌ها، و تاپل‌ها، امکانات بیشتری برای پیاده‌سازی الگوریتم‌های پیچیده به برنامه‌نویسان ارائه می‌دهند. به طور خلاصه، توابع درونی پایتون ابزاری قدرتمند و منعطف برای هر سطحی از برنامه‌نویسی محسوب می‌شوند که به برنامه‌نویسان کمک می‌کنند تا کدی موثرتر و حرفه‌ای‌تر بنویسند.

فهرستی از توابع درونی پایتون را به همراه توضیحات و آرگومان‌های مربوط به هر تابع در جدول ۱-۷ ارائه شده است. این توابع جزو توابع اصلی پایتون هستند و در هر برنامه‌ای بدون نیاز به وارد کردن کتابخانه خاصی قابل استفاده‌اند.

جدول ۱-۷ - جدول مربوط به توابع درونی موجود در پایتون

توضیحات	توابع و آرگومان‌ها
مقدار مطلق عدد x را بازمی‌گرداند.	$\text{abs}(x)$

بررسی می کند که آیا تمام مقادیر موجود در iterable درست هستند یا خیر

all(iterable)

(همه برابر True)

بررسی می کند که آیا حداقل یکی از مقادیر موجود در iterable درست است

any(iterable)

یا خیر.

نسخه ای قابل نمایش از شیء object بر می گرداند که فقط شامل کاراکترهای

ascii(object)

ASCII باشد.

رشته ای از نمایش دودویی (باینری) عدد صحیح x را بازمی گرداند.

bin(x)

مقدار x را به مقدار بولین (False یا True) تبدیل می کند.

bool([x])

آرایه ای از بایت ها بر اساس source ورودی ایجاد می کند.

bytearray([source,
encoding, errors])

شیء بایتی از source ورودی ایجاد می کند.

bytes([source, encoding,
errors])

بررسی می کند که آیا object قابل فراخوانی (مثل توابع) است یا خیر.

callable(object)

کاراکتر مربوط به مقدار عددی i در جدول Unicode را بازمی گرداند.

chr(i)

یک عدد مختلط ایجاد می کند.

complex([real, imag])

ایجاد نوع داده دیکشنری.

dict(**kwargs))

تقسیم و باقیمانده تقسیم a و b را به صورت زوج (q, r) بر می گرداند.

divmod(a, b)

شیء enumerate از iterable ایجاد می کند که اندیس ها را نیز بر می گرداند.

enumerate(iterable,
start=0)

مقادیری از iterable که تابع function برایشان True بر می گرداند، انتخاب

filter(function, iterable)

می کند.

عدد x را به یک عدد اعشاری تبدیل می کند.

float([x])

قالب‌بندی <code>value</code> طبق <code>format_spec</code>	<code>format(value, format_spec)</code>
یک مجموعه (set) تغییرناپذیر از عناصر <code>iterable</code> می‌سازد.	<code>frozenset([iterable])</code>
مقدار ویژگی <code>name</code> در شیء <code>object</code> را بازمی‌گرداند.	<code>getattr(object, name, [default])</code>
دیکشنری‌ای از متغیرهای سطح گلوبال برنامه را بازمی‌گرداند.	<code>globals()</code>
بررسی می‌کند که آیا شیء <code>object</code> دارای ویژگی <code>name</code> است یا خیر.	<code>hasattr(object, name)</code>
مقدار هش (hash) شیء <code>object</code> را بازمی‌گرداند.	<code>hash(object)</code>
نمایش هگزا (هگزادسیمال) عدد <code>x</code> را به صورت رشته بازمی‌گرداند.	<code>hex(x)</code>
شناسه یکتای شیء <code>object</code> در حافظه را بازمی‌گرداند.	<code>id(object)</code>
ورودی را از کاربر دریافت و آن را به صورت رشته بازمی‌گرداند.	<code>input([prompt])</code>
عدد <code>x</code> را به یک عدد صحیح (integer) تبدیل می‌کند.	<code>int([x, base])</code>
بررسی می‌کند که آیا شیء <code>object</code> نمونه‌ای از کلاس <code>classinfo</code> است یا خیر.	<code>isinstance(object, classinfo)</code>
طول دنباله یا تعداد عناصر موجود در <code>s</code> را بازمی‌گرداند.	<code>len(s)</code>
ایجاد نوع داده‌ی فهرست.	<code>list([iterable])</code>
دیکشنری‌ای از متغیرهای محلی تابع فعلی را بازمی‌گرداند.	<code>locals()</code>
تابع <code>function</code> را بر روی همه عناصر <code>iterable</code> اعمال و نتایج را بازمی‌گرداند.	<code>map(function, iterable, ...)</code>
بزرگ‌ترین مقدار موجود در <code>iterable</code> را بازمی‌گرداند.	<code>max(iterable)</code>
کوچک‌ترین مقدار موجود در <code>iterable</code> را بازمی‌گرداند.	<code>min(iterable)</code>
عنصر بعدی <code>iterator</code> را بازمی‌گرداند.	<code>next(iterator, default)</code>

نمایش اکتال (بر مبنای ۸) عدد x را به صورت رشته بازمی‌گرداند.	oct(x)
فایل مشخص شده در file را باز می‌کند.	open(file, mode='r', ...)
کد Unicode کاراکتر c را بازمی‌گرداند.	ord(c)
محاسبه base به توان exp.	pow(base, exp)
چاپ objects با جداکننده sep و پایان‌دهنده .end.	sep=' ', print(objects, end='\n', ...)
تولید یک دنباله عددی از start تا stop با فاصله‌های step.	range(start, stop[, step])
نمایش رسمی شیء object که قابل اجراست، بازمی‌گرداند.	repr(object)
دنباله‌ای معکوس از seq را بازمی‌گرداند.	reversed(seq)
گرد کردن عدد number به تعداد اعشار ndigits.	round(number[, ndigits])
ایجاد نوع داده‌ی مجموعه.	set([iterable])
دنباله‌ای مرتب شده از iterable را بازمی‌گرداند.	sorted(iterable)
ایجاد نوع داده‌ی رشته.	str(object)
جمع عناصر موجود در iterable با مقدار اولیه start.	sum(iterable[, start=0])
ایجاد نوع داده‌ی تاپل.	tuple([iterable])
نوع شیء object را بازمی‌گرداند یا یک کلاس جدید ایجاد می‌کند.	type(object) type(name, bases, dict)
شیء zip که عناصر iterables را به صورت زوج‌های مرتب ترکیب می‌کند، بازمی‌گرداند.	zip(iterables)

جدول ۱-۷ نمای کلی از توابع درونی پایتون به همراه آرگومان‌های معمولشان را نشان می‌دهد. برای جزئیات بیشتر درباره هر کدام، می‌توانید از تابع `help()` استفاده کنید.

۷-۷) آشنایی با `lambda` در زبان برنامه‌نویسی پایتون

در زبان برنامه‌نویسی پایتون، "عبارات `lambda`" برای تعریف توابع بدون نام و مختصر به کار می‌روند. این عبارات به گونه‌ای طراحی شده‌اند که به سادگی و با سرعت بتوان توابع کوتاهی را بدون استفاده از کلمه‌ی کلیدی `'def'` تعریف کرد، که اغلب تنها برای استفاده در مکان‌های خاصی از کد مورد نیاز هستند.

یک عبارت `'lambda'` از ساختاری بسیار ساده بهره می‌برد: کلمه‌ی کلیدی `'lambda'` به دنبال یک یا چند آرگومان ورودی(`arguments`) و سپس "یک عبارت(`expression`)" که نتیجه‌ی محاسبه‌ی تابع است، می‌آید. به عبارت دیگر، ساختار آن به شکل زیر است:

lambda arguments: expression

کاربردهای عبارت‌های `'lambda'` اغلب در مواقعی است که نیاز به استفاده از توابع کوچک و موقت در کنار توابعی چون `'map()', filter(), sorted()'` وجود دارد.

۱-۷-۷) مثال‌های مربوط به `lambda`

مثال ۱. با استفاده از `lambda` توان دوم هر عددی را از فهرستی بدست آورید.

python example - functions.py

```
283 # Functions in Python (توابع در پایتون)  
284  
285 # ** Lambda Function ** توابع لامدا  
286  
287 # Example 1: توان دوم اعداد فهرست  
288 numbers = [1, 2, 3, 4, 5]  
289 squared = list(map(lambda x: x**2, numbers))  
290  
291 print(squared)
```

شکل ۷-۲۲- مثال مربوط به توان دوم اعداد یک فهرست با استفاده از *lambda*

خروجی شکل ۷-۲۲:

[1, 4, 9, 16, 25]

توضیحات شکل ۷-۲۲:

این قطعه کد از تابع درونی `map` و عبارت `lambda` استفاده می‌کند تا توان دوم تمام اعداد موجود در فهرست `numbers` را محاسبه کند. سپس نتیجه به فهرستی جدید تبدیل شده و چاپ می‌شود.

۱. تعریف فهرست اعداد اولیه:

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

این خط فهرستی شامل اعداد صحیح ۱ تا ۵ را ایجاد می‌کند.

۲. محاسبه‌ی توان دوم با استفاده از `map` و `lambda`:

```
squared = list(map(lambda x: x**2, numbers))
```

▪ : یک عبارت `lambda` تعریف شده که هر عدد ورودی `x` را به توان دوم

می‌رساند.

▪ `map` : این تابع، عبارت `lambda` را روی هر عنصر از فهرست `numbers` اعمال می‌کند.

▪ `list()` : خروجی تابع `map` یک شیء قابل پیمایش (`map object`) است. برای تبدیل آن به

فهرست، از تابع `list()` استفاده شده است.

۳. چاپ نتیجه:

```
print(squared)
```

این خط، فهرست جدید حاوی توان دوم هر عدد از فهرست اولیه را چاپ می‌کند.

❖ عدد `1` به توان دوم: ($1^2 = 1$)

❖ عدد `2` به توان دوم: ($2^2 = 4$)

❖ عدد `3` به توان دوم: ($3^2 = 9$)

❖ عدد `4` به توان دوم: ($4^2 = 16$)

❖ عدد `5` به توان دوم: ($5^2 = 25$)

مثال ۲. با استفاده از `lambda`، مقادیر دو فهرست را با هم دیگر جمع کنید.

python example - functions.py

```
295 # Functions in Python (توابع در پایتون)  
296  
297 # ** Lambda Function **  
298 # Example 2: جمع کردن مقادیر دو فهرست متناظر  
299  
300 list1 = [1, 2, 3]  
301 list2 = [4, 5, 6]  
302 summed = list(map(lambda x, y: x + y, list1, list2))  
303  
304 print(f"Sum of the Lists: {summed}")
```

شکل ۷-۲۳- مثال مربوط به محاسبه مجموع دو فهرست متناظر با استفاده از *lambda*

خروجی شکل ۷-۲۳:

Sum of the Lists: [5, 7, 9]

توضیحات شکل ۷-۲۳:

این قطعه کد از تابع `map` و عبارت `lambda` برای جمع کردن عناصر متناظر دو فهرست `list1` و `list2` استفاده می‌کند. نتیجه‌ی این عملیات به صورت یک فهرست جدید ذخیره و چاپ می‌شود.

تحلیل کد:

۱. تعریف دو فهرست اولیه:

list1 = [1,2,3]

list2 = [4,5,6]

این دو فهرست شامل سه عدد صحیح هستند که عناصر آن‌ها متناظر با هم جمع خواهند شد.

۲. جمع کردن عناصر متناظر با `map` و `lambda` :

```
summed = list(map(lambda x, y: x + y, list1, list2))
```

: یک عبارت `lambda` تعریف شده که دو عدد ورودی `x` و `y` را با هم جمع می‌کند.

: این تابع، عبارت `lambda` را به طور موازی روی عناصر متناظر دو فهرست اعمال می‌کند:

- عنصر اول از `list1` با عنصر اول از `list2` جمع می‌شود.

- عنصر دوم از `list1` با عنصر دوم از `list2` جمع می‌شود.

و به همین ترتیب.

: خروجی `map` که یک شیء قابل پیمایش (`map object`) است، به یک فهرست تبدیل می‌شود.

۳. چاپ نتیجه:

```
print(summed)
```

$1 + 4 = 5 \triangleright$

$2 + 5 = 7 \triangleright$

$3 + 6 = 9 \triangleright$

نکات:

- ✓ این روش بسیار مفید است برای زمانی که نیاز دارید دو فهرست یا بیشتر را به صورت موازی پردازش کنید، بدون نیاز به نوشتن حلقه‌های اضافی. تابع `map` همراه با `lambda` کدی خواناتر و فشرده‌تر تولید می‌کند.

مثال ۳. با استفاده از `lambda`، فهرست میوه‌ها را براساس تعداد حروفشان مرتب‌سازی کنید.

python example - functions.py

```
308 # Functions in Python (تابع در پایتون)  
309  
310 # ** Lambda Function **  
311 # Example 3: مرتب‌سازی لیستی از رشته‌ها بر اساس طول آن‌ها  
312  
313 words = ['apple', 'banana', 'kiwi', 'grape']  
314 sorted_words = sorted(words, key=lambda x: len(x))  
315  
316 print(f"Sorted List: {sorted_words}")
```

شکل ۷-۲۴-مثال مربوط به مرتب‌سازی عناصر یک فهرست با توجه به طول عنصر با استفاده از `lambda`

خروجی شکل ۷-۲۴:

Sorted List: ['kiwi', 'apple', 'grape', 'banana']

توضیحات شکل ۷-۲۴:

این کد از تابع درونی `sorted` و عبارت `lambda` برای مرتب‌سازی فهرست `words` بر اساس طول رشته‌ها استفاده می‌کند. نتیجه‌ی این مرتب‌سازی به ترتیب صعودی طول رشته‌ها ذخیره و چاپ می‌شود.

۱. تعریف فهرست اولیه:

`words = ['apple', 'banana', 'kiwi', 'grape']`

فهرستی شامل چند رشته (کلمات) است که قرار است بر اساس طول آن‌ها مرتب شود.

۲. مرتبسازی با استفاده از `sorted` و `lambda` :

```
sorted_words = sorted(words, key=lambda x: len(x))
```

▪ `sorted` : تابعی است که فهرستی مرتب شده بر اساس یک معیار خاص بازمی‌گرداند، بدون اینکه فهرست

اصلی تغییر کند.

▪ `key` : این پارامتر مشخص می‌کند که چگونه عناصر فهرست باید مقایسه شوند.

▪ `lambda` : یک عبارت `lambda x: len(x)` است که برای هر عنصر `x` از فهرست، طول آن رشته را

بازمی‌گرداند. این مقدار طول به عنوان معیار مرتبسازی استفاده می‌شود.

▪ نتیجه: رشته‌ها بر اساس طولشان به ترتیب صعودی مرتب می‌شوند.

۳. چاپ نتیجه:

```
print(sorted_words)
```

این خط، فهرست مرتب شده را چاپ می‌کند.

توضیح مرتبسازی:

▪ `kiwi` : طول ۴ کاراکتر

▪ `grape` : طول ۵ کاراکتر

▪ `apple` : طول ۵ کاراکتر

▪ `banana` : طول ۶ کاراکتر

رشته‌هایی با طول یکسان (مانند `apple` و `grape`) بر اساس ترتیب اولیه‌شان در فهرست مرتب می‌شوند (چون `sorted` پایدار است).

نکات:

می‌توانید با تغییر `key`، معیارهای دیگری برای مرتبسازی تعریف کنید.

برای مرتبسازی نزولی، از پارامتر `reverse=True` استفاده کنید:

```
sorted_words = sorted(words, key=lambda x: len(x), reverse=True)
```

```
print(sorted_words)
```

به طور کلی، عبارات `lambda` ابزاری قدرتمند برای بهینه‌سازی کد و حذف تعریف توابع غیرضروری در موقع خاص هستند؛ اما برای توابع پیچیده و طولانی توصیه می‌شود از توابع معمولی با کلمه‌ی کلیدی `def` استفاده شود تا کد همچنان خوانایی و قابلیت نگهداری بالایی داشته باشد.

۷-۸) آشنایی با توابع درونی `map()`, `filter()`, `zip()` در پایتون

۷-۸-۱) تابع درونی `zip()`

تابع درونی `zip` در پایتون یکی از توابع کاربردی برای ترکیب عناصر چندین "دنباله" (مانند فهرست‌ها، تاپل‌ها و غیره) به صورت "زوج‌های مرتب" است. این تابع با قرار دادن عناصر مربوطه از هر دنباله در کنار هم، "شیء" "zip" جدیدی ایجاد می‌کند که می‌تواند به صورت یک دنباله از "تاپل‌ها" پیمایش شود.

ساختار تابع `zip` به صورت زیر است:

`zip(*iterables)`

در این ساختار، `iterables` می‌تواند شامل هر تعداد دنباله (مانند فهرست‌ها و تاپل‌ها) باشد. تابع `zip` به طور پیش‌فرض تا جایی عناصر را ترکیب می‌کند که کوتاه‌ترین دنباله به پایان برسد، یعنی اگر دنباله‌ها طول‌های متفاوتی داشته باشند، از طول کوتاه‌ترین دنباله پیروی می‌شود.

مثال ساده‌ی زیر نحوه‌ی عملکرد این تابع را نشان می‌دهد:

python example - functions.py

```
320 # Functions in Python (توابع در پایتون)
321
322 # zip(), map(), filter()
323 # Zip Function ( zip() )
324
325 names = ['Ali', 'Sara', 'Reza']
326 ages = [30, 27, 24]
327 combined = zip(names, ages)
328
329 print(list(combined))
```

شکل ۷-۲۵- مثال مربوط به الحاق دو فهرست به صورت جفت با استفاده از تابع وابسته‌ی `zip`

خروجی شکل ۷-۲۵:

`I('Ali', 30), ('Sara', 27), ('Reza', 24)]`

توضیحات شکل ۷-۲۵:

هر عنصر از `names` با عنصر متناظر در `ages` ترکیب شده و یک تاپل تشکیل داده است.

تابع `zip` به ویژه در موقعی مفید است که نیاز به پردازش موازی داده‌ها از چند دنباله وجود دارد و می‌خواهیم عناصر متناظر را به صورت جفتی در کنار هم نگهداری و مدیریت کنیم. اگر نیاز باشد که خروجی تابع `zip` به جای یک شیء قابل پیمایش به یک نوع داده‌ی مشخص مانند فهرست یا دیکشنری تبدیل شود، می‌توان از توابع `dict(zip(...))` و `list(zip(...))` استفاده کرد.

۲-۸-۷) تابع درونی `map()`

تابع درونی `map` در پایتون ابزاری قدرتمند و پرکاربرد است که به شما امکان می‌دهد یک "عملکرد مشخص" را روی تمام عناصر یک "دنباله" (مانند فهرست، تاپل، یا هر نوع iterable دیگر) به صورت همزمان اعمال کنید و نتایج را در قالب یک دنباله جدید دریافت کنید. این تابع برای موقعی ایده‌آل است که می‌خواهید یک عملیات مشابه (مانند محاسبات ریاضی، تبدیل نوع، یا هر عملیات دیگر) را بر روی تمامی عناصر یک دنباله اعمال کنید، بدون آن‌که نیاز به نوشتن حلقه باشد.

ساختار تابع `map` به صورت زیر است:

map(function, iterable)

در این ساختار، `function` تابعی است که می‌خواهید روی عناصر `iterable` اعمال کنید. این تابع باید به عنوان ورودی، یک عنصر از دنباله را دریافت کرده و یک نتیجه را برگرداند.

به عنوان مثال، فرض کنید می‌خواهید فهرستی از اعداد را به توان دو برسانید. می‌توانید از `map` استفاده کنید تا تابع `lambda` این کار را بر روی همه عناصر اعمال کند:

python example - functions.py

```
333 # Functions in Python (توابع در پایتون)
334
335 # zip(), map(), filter()
336 # Map Function ( map() )
337
338 numbers = [1, 2, 3, 4]
339 squared_numbers = map(lambda x: x**2, numbers)
340
341 print(list(squared_numbers))
```

شکل ۷-۲۶- مثال مربوط به به توان دو رساندن عناصر یک فهرست با استفاده از تابع وابسته‌ی *map*

خروجی شکل ۷-۲۶:

[1, 4, 9, 16]

توضیحات شکل ۷-۲۶:

تابع `map` برای هر عنصر در `numbers`، تابع `lambda x: x**2` را اجرا کرده و نتیجه را به عنوان یک شیء قابل پیمایش (به شکل `map object`) برمی‌گرداند. برای مشاهده‌ی نتایج، معمولاً آن را به یک نوع داده‌ی مشخص، مانند فهرست، تبدیل می‌کنند.

از ویژگی‌های دیگر `map`، "بهینه‌سازی و سرعت بیشتر" آن نسبت به حلقه‌های معمولی در پردازش‌های ساده است، به خصوص زمانی که با دنباله‌های بزرگ سروکار داریم. همچنین، تابع `map` می‌تواند چندین دنباله را به صورت موازی بپذیرد، به شرط آنکه تابع ارائه شده، به تعداد آرگومان‌های ورودی دنباله‌ها طراحی شده باشد.

python example - functions.py

```
345 # Functions in Python (توابع در پایتون)
346
347 # zip(), map(), filter()
348 # Map Function ( map() )
349
350 numbers1 = [1, 2, 3]
351 numbers2 = [4, 5, 6]
352 added_numbers = map(lambda x, y: x + y, numbers1, numbers2)
353
354 print(list(added_numbers))
```

شکل ۷-۷- مثال مربوط به جمع اعداد دو فهرست به صورت جفت با استفاده از تابع وابسته‌ی *map*

خروجی شکل ۷-۷:

[5, 7, 9]

توضیحات شکل ۷-۷:

در اینجا، هر عنصر از `numbers1` با عنصر متناظر خود در `numbers2` جمع شده است.

به طور کلی، تابع `map` ابزاری بسیار مفید در پایتون است که به شما امکان می‌دهد کدی ساده‌تر و بهینه‌تر

برای عملیات تکراری بنویسید و کدتان را خواناتر و مرتب‌تر کنید.

۳-۸-۷ تابع درونی filter()

تابع درونی `filter` در پایتون ابزاری کاربردی برای "فیلتر کردن عناصر" یک "دنباله" (مانند فهرست، تاپل یا مجموعه) بر اساس یک شرط مشخص است. این تابع برای موقعی ایدهآل است که می‌خواهد تنها "عناصری" از یک دنباله را که به "شرطی خاص" پایبند هستند، استخراج کنید و بقیه را حذف کنید.

ساختار تابع `filter` به صورت زیر است:

filter(function, iterable)

در این ساختار، `function` تابعی است که هر عنصر از `iterable` را به عنوان ورودی دریافت می‌کند و "یک مقدار بولین (True یا False)" بازمی‌گرداند. عناصری از `iterable` که `function` برای آن‌ها مقدار True بازمی‌گرداند، در نتیجه‌ی نهایی قرار می‌گیرند؛ بقیه عناصر حذف خواهند شد. در صورت استفاده از `None` به جای None، تابع `filter` تنها عناصری که خودشان به طور طبیعی برابر با True هستند (یعنی مقدار صفر، False و None نیستند) را نگه می‌دارد.

به عنوان مثال، اگر بخواهید از یک فهرست، اعداد زوج و فرد را جدا کنید، می‌توانید از تابع `filter` با استفاده از تابع `lambda` با توجه به شکل ۲۸-۷ استفاده کنید:

python example - functions.py

```
358 # Functions in Python (توابع در پایتون)
359
360 # zip(), map(), filter()
361 # Filter Function ( filter() )
362
363 numbers = [1, 2, 3, 4, 5, 6]
364 even_numbers = filter(lambda x: x % 2 == 0, numbers)
365 odd_numbers = filter(lambda x: x%2 != 0, numbers)
366
367 print(f"Even numbers: {list(even_numbers)}")
368 print(f"Odd numbers: {list(odd_numbers)}")
```

شکل ۷-۲۸- مثال مربوط به جداسازی اعداد زوج و فرد با استفاده از تابع وابسته *filter*

خروجی شکل ۷-۲۸:

Even numbers: [2, 4, 6]

Odd numbers: [1, 3, 5]

توضیحات مربوط به شکل ۷-۲۸:

در `even_numbers`، تابع `filter`، تنها آن عناصری از `numbers` را انتخاب کرد که شرط `x % 2 == 0` برایشان برقرار است، یعنی اعداد زوج.

در `odd_numbers`، تابع `filter`، تنها آن عناصری از `numbers` را انتخاب کرد که شرط `x % 2 != 0` برایشان برقرار است، یعنی اعداد فرد.

تابع `filter` به طور معمول با توابعی چون `lambda` یا توابع تعریف شده دیگر ترکیب می‌شود تا فیلترینگ داده‌ها را ساده و قابل تنظیم کند. هم‌چنین این تابع، یک شیء `filter object` بازمی‌گرداند که می‌توان آن را به انواع داده‌های دیگر مانند فهرست، تاپل یا دیکشنری تبدیل کرد تا نتیجه به راحتی قابل دسترس باشد.

به‌طور کلی، تابع `filter` ابزار قدرتمندی برای پالایش و انتخاب داده‌ها در مجموعه‌های بزرگ است و به‌ویژه در پردازش داده‌ها و ایجاد زیرمجموعه‌هایی که با شرایط خاصی همخوانی دارند، بسیار کاربرد دارد.

فصل هشتم

خطاهای در زبان برنامه‌نویسی پایتون

۸) خطاهای و انواع آن‌ها در زبان برنامه‌نویسی پایتون

در فرآیند توسعه نرم‌افزار، خطاهای^{۶۷} بخش جدایی‌ناپذیری از برنامه‌نویسی هستند. پایتون، به عنوان یک زبان سطح بالا، ابزارهای متعددی برای شناسایی و مدیریت خطاهای ارائه می‌دهد. این خطاهای معمولاً^{۶۸} به سه دسته اصلی تقسیم می‌شوند: خطاهای ساختاری^{۶۹}، خطاهای زمان اجرا^{۷۰} و خطاهای منطقی^{۷۱}. در ادامه، به بررسی هر یک از این خطاهای را پرداخته می‌شود.

۸-۱) خطاهای ساختاری

این دسته از خطاهای زمانی رخ می‌دهند که قوانین گرامری و ساختاری و نحوی زبان پایتون رعایت نشود. پایتون هنگام اجرا، کد را تفسیر می‌کند و در صورت مواجهه با خطاهای ساختاری، برنامه پیش از اجرا متوقف می‌شود. شکل ۸-۱، نمونه‌ای از خطای ساختاری می‌باشد.

```
python example - Errors_Handling.py
1 # Errors in Python (خطاهای در پایتون)
2
3 # Types of Errors
4   # Syntax Errors
5   # Runtime Errors
6   # Logical Errors
7
8
9 # Types of Errors
10  # Syntax Errors
11
12 print("Hello World")
```

شکل ۸-۱- مثالی از خطای ساختاری در پایتون

⁶⁷ Errors

⁶⁸ Syntax Errors

⁶⁹ Runtime Errors

⁷⁰ Logical Errors

خروجی شکل ۱-۸:

SyntaxError: '(' was never closed

توضیحات شکل ۱-۸:

در اینجا پرانتز بسته فراموش شده است، که منجر به خطای ساختاری می‌شود.

۲-۸) خطاهای زمان اجرا

این نوع خطاهای در زمان اجرای برنامه رخ می‌دهند. برخلاف خطاهای ساختاری، برنامه تا رسیدن به خطای موردنظر اجرا می‌شود، اما در آن لحظه متوقف می‌گردد.

مثال ۱. نمونه‌ای از مثال رایج، طبق شکل ۲-۸ تقسیم یک عدد بر صفر می‌باشد.

python example - Errors_Handling.py

```
18 # Types of Errors
19 # Syntax Errors
20 # Runtime Errors
21 # Logical Errors
22
23 # Runtime Errors
24
25 print(100/0)
```

شکل ۲-۸-۱- مثالی از خطای زمان اجرا با تقسیم عددی بر صفر

خروجی شکل ۲-۸:

ZeroDivisionError: division by zero

مثال ۲. دسترسی به شاخص نامعتبر در فهرست طبق شکل ۳-۸ می‌تواند موجب خطای زمان اجرا گردد.

python example - Errors_Handling.py

```
29 # Errors in Python (خطاهای در پایتون)
30
31 # Types of Errors
32 # Syntax Errors
33 # Runtime Errors
34 # Logical Errors
35
36 # Runtime Errors
37
38 my_list = [1, 2, 3]
39 print(my_list[5])
```

شکل ۳-۸- مثالی از خطای زمان اجرا با دادن اندیس غیرمجاز

خروجی شکل ۳-۸:

IndexError: list index out of range

۳-۸) خطاهای منطقی

این خطاهای زمانی رخ می‌دهند که برنامه بدون توقف اجرا می‌شود، اما نتیجه‌ای که تولید می‌کند، نادرست است.

خطاهای منطقی معمولاً ناشی از اشتباهات برنامه‌نویس در طراحی منطق کد هستند.

برای مثال، با توجه به شکل ۴-۸، فرض کنید قصد محاسبه میانگین دو عدد را دارید.

python example - Errors_Handling.py

```
43 # خطاهای در پایتون (Errors in Python)
44
45 # Types of Errors
46 # Syntax Errors
47 # Runtime Errors
48 # Logical Errors
49
50 # Logical Errors
51 def average(a, b):
52     return a + b / 2
53     به صورت زیر محاسبه خواهد کرد
54     # a + (b/2)
55 print(f"First Ans: {average(a=14,b=12)}")
```

شکل ۴-۸- مثالی از خطای منطقی به دلیل عدم رعایت اولویت‌بندی عملگرها

خروجی شکل ۴-۸:

First Ans: 20.0

برای حل خطای شکل ۴-۸، باید اولویت‌بندی عملگرها را طبق شکا ۵-۸ رعایت کنید.

python example - Errors_Handling.py

```
43 # Errors in Python (خطاهای در پایتون)
44
45 # Types of Errors
46 # Syntax Errors
47 # Runtime Errors
48 # Logical Errors
49
50 # Logical Errors خطای منطقی
51 def average(a, b):
52     return a + b / 2
53     به صورت زیر محاسبه خواهد کرد
54     # a + (b/2)
55 print(f"First Ans: {average(a=14,b=12)}")
56
57 # Solve Problem حل خطای منطقی
58 def average(a, b):
59     return (a + b) / 2
60     به صورت زیر محاسبه خواهد کرد #
61     # (a + b) / 2
62 print(f"Second Ans: {average(a=14,b=12)}")
```

شکل ۸-۵- مثالی از خطای منطقی و حل آن

خروجی شکل ۸-۵:

First Ans: 20.0

Second Ans: 13.0

۴-۸) سایر خطاهای متداول در پایتون

پایتون به دلیل غنی بودن در مدیریت خطا، مجموعه‌ای از خطاهایی از خطاهایی از پیش تعریف شده را ارائه می‌دهد که برخی از آن‌ها عبارتند از:

. زمانی که عملیاتی بر روی نوع داده نامناسبی انجام شود. **TypeError** ♦

python example - Errors_Handling.py

```
66 # Errors in Python (خطاهای در پایتون)
67 # Other Errors
68 #    TypeErrors
69
70 def Sum_Operation(a, b):
71     return f"Sum of a , b: {a + b}"
72
73 print(Sum_Operation("12", 4))
```

شکل ۴-۸- مثالی از خطای *TypeError* در پایتون

خروجی شکل ۴-۸

TypeError: can only concatenate str (not "int") to str

. زمانی که مقدار داده ورودی برای یک عملیات مناسب نباشد. به شکل ۷-۸ توجه نمایید. **ValueError** ♦

python example - Errors_Handling.py

```
77 # Errors in Python (خطاهای در پایتون)
78 # Other Errors
79 #    ValueError
80
81 def Value_Error_Example(a):
82     return f"{int(a)}"
83
84 print(Value_Error_Example("Hello World!"))
```

شکل ۸-۷- مثالی از خطای *ValueError* در پایتون

خروجی شکل ۸-۷:

ValueError: invalid literal for int() with base 10: 'Hello World!'

زمانی که به یک متغیر یا نام تعریف نشده دسترسی پیدا کنید. به شکل ۸-۸ توجه نمایید. **NameError** ♦♦♦

python example - Errors_Handling.py

```
88 # Errors in Python (خطاهای در پایتون)
89 # Other Errors
90 #    NameError
91
92 هیچ متغیری یا تابعی تعریف نکردیم
93 print(f"value of x is {x}")
```

شکل ۸-۸- مثالی از خطای *NameError* در پایتون

NameError: name 'x' is not defined

زمانی که کلیدی وجود ندارد اما از دیکشنری درخواست شود. به شکل ۹-۸ توجه نمایید. **KeyError** ♦♦♦

python example - Errors_Handling.py

```
97 # Errors in Python (خطاهای در پایتون)
98 # Other Errors
99 #    KeyErrors
100
101 mydict = {
102     "firstName": "Reza",
103     "age": 23
104 }
105 # Available Key
106 print(f"First Name:{mydict["firstName"]}")
107
108 # Not Available Key
109 print(f"Last Name:{mydict["lastName"]}")
```

شکل ۹-۸-۹- مثالی از خطای *keyError* در پایتون

First Name:Reza

Traceback (most recent call last):

print(f"Last Name:{mydict["lastName"]}")

~~~~~^~~~~~^~~~~~^~~~~~^~~~~~^~~~~~^

## **KeyError: 'lastName'**

---

زمانی که یک مازول یا نام در مازول یافت نشود. به شکل ۱۰-۸ توجه نمایید.

```
python example - Errors_Handling.py

113 # Errors in Python (خطاهای در پایتون)
114 # Other Errors
115 # ImportErrors
116
117 import no_exist_module
118
119 print(help(no_exist_module))
```

شکل ۱۰-۸- مثالی از خطای **ImportError** در پایتون

خروجی شکل ۱۰-۸ :

---

## **ModuleNotFoundError: No module named 'nonexistent\_module'**

---

نکته: شناخت انواع خطاهای و نحوه مدیریت آن‌ها یکی از مهارت‌های اساسی برای یک برنامه‌نویس حرفه‌ای است. پایتون ابزارهای فراوانی برای کمک به برنامه‌نویسان در این زمینه فراهم کرده است، از جمله مدیریت خطاهای استفاده از ابزارهای خطایابی<sup>۷۱</sup>.

---

<sup>۷۱</sup> Debugging

# فصل نهم

مدیریت خطاها در زبان برنامه‌نویسی پایتون

## ۹) مدیریت خطاها در پایتون

در برنامهنویسی، ممکن است خطاها یا شرایط غیرمنتظره‌ای رخ دهند که در صورت عدم مدیریت مناسب، باعث متوقف شدن اجرای برنامه شوند. زبان برنامهنویسی پایتون ابزار قدرتمندی به نام **مدیریت خطاها**<sup>۷۲</sup> ارائه می‌دهد تا این مشکلات را به شیوه‌ای ایمن و ساخت‌یافته کنترل کنید. ساختار اصلی مدیریت استثناهای در پایتون از کلمات کلیدی **try** و **except** استفاده می‌کند.

### ۱-۹) ساختار **try/except** برای مدیریت کردن خطاها

ساختار پایه‌ای استفاده از **try** و **except** همانند شکل ۱-۹ می‌باشد.

```
python example - Errors_Handling.py

123 # Errors in Python (خطاها در پایتون)
124 # Errors Handling (مدیریت خطاها)
125 # Try/Except
126 # Try/Except/Else/Finally
127
128 # Try/Except
129 """
130 try:
131     کدی که ممکن است خطا تولید کند #
132 except ErrorType:
133     کدی که در صورت رخدان خطا اجرا می‌شود #
134
135 """
```

شکل ۱-۹- ساختار پایه‌ای **try/except** در پایتون

- بخش **try**: در این بلوک، کدی قرار می‌گیرد که ممکن است خطایی در حین اجرا تولید کند.

<sup>72</sup> Exception Handling

• بخش except: اگر خطایی در بلوک try رخ دهد، اجرای برنامه متوقف نمی‌شود. در عوض، کنترل به بلوک

except منتقل می‌شود و شما می‌توانید پاسخی مناسب برای آن خطا ارائه دهید.

در شکل ۲-۹، برنامه تلاش می‌کند یک عدد را بر صفر تقسیم کند اما خطای ZeroDivisionError ایجاد می‌کند:

python example - Errors\_Handling.py

```
138 # Errors in Python (خطاهای در پایتون)
139 # Errors Handling (مدیریت خطاهای)
140 # Try/Except
141
142 try:
143     result = 265 / 0
144 except ZeroDivisionError:
145     print("Division by zero is not possible.")
```

شکل ۲-۹-۲- مثالی از مدیریت خطای ZeroDivisionError در پایتون

خروجی شکل ۲-۹:

---

*Division by zero is not possible.*

---

توضیحات شکل ۲-۹:

در اینجا، به جای متوقف شدن برنامه، خطا شناسایی شده و پیام مناسبی نمایش داده می‌شود.

## ۲-۹) مدیریت چندین نوع خطا

همانطور که در شکل ۳-۹ مشاهده می‌کنید، می‌توانید چندین بلوک except برای مدیریت انواع مختلف خطاهای تعریف کنید.

python example - Errors\_Handling.py

```
149 # Errors in Python (خطاهای در پایتون)  
150 # Errors Handling خطاهای مدیریت  
151     # Try/Except  
152  
153 # Many Except Section  
154 try:  
155     num = int(input("Enter a number:"))  
156     result = 10 / num  
157 except ValueError:  
158     print("Please Enter Number Not Integer!")  
159 except ZeroDivisionError:  
160     print("Division by zero is not possible!")
```

شکل ۳-۹- مثالی از مدیریت چندین نوع خطا در پایتون

خروجی شکل ۳-۹:

اگر در ورودی مقدار Hello را قرار دهید، خروجی که می‌گیرید به این صورت خواهد بود:

---

*Enter a number:Hello*

*Please Enter Number Not Integer!*

---

اگر در ورودی عدد صفر را قرار دهید، خروجی که می‌گیرید به این صورت خواهد بود:

---

*Enter a number:0*

*Division by zero is not possible!*

---

### ۴-۹) استفاده از بلوک عمومی **except**

اگر نوع خطا مشخص نباشد، می‌توانید طبق شکل ۴-۹ از بلوک عمومی **except** استفاده کنید:

python example - Errors\_Handling.py

```
164 # Errors in Python (خطاهای در پایتون)  
165 # Errors Handling (مدیریت خطاهای)  
166 # Try/Except  
167  
168 # If Error is not clear:  
169 try:  
170     num = int("hello")  
171 except:  
172     print("An unspecified error has occurred.")
```

شکل ۴-۹-۴-مثالی از استفاده بلوک عمومی **except** برای زمانی که نوع خطا مشخص نباشد

خروجی شکل ۴-۹:

---

*An unspecified error has occurred.*

---

## ۴-۹) ترکیب با **finally** و **else**

برای مدیریت بهتر، می‌توانید از دو بخش **else** و **finally** نیز استفاده کنید:

- بخش **else**: اگر هیچ خطایی در بلوک **try** رخ ندهد، این بخش اجرا می‌شود.
  - بخش **finally**: این بخش همواره اجرا می‌شود، چه خطایی رخ دهد و چه رخ ندهد.
- برای درک بهتر دو بخش **else** و **finally** به شکل ۵-۹ توجه نمایید.

python example - Errors\_Handling.py

```
176 # Errors in Python (خطاهای در پایتون)
177 # Errors Handling (مدیریت خطاهای)
178 # Try/Except/Else/Finally
179
180 try:
181     num = int(input("Enter a number: "))
182     result = 10 / num
183 except ZeroDivisionError:
184     print("Division by zero is not possible!")
185 except ValueError:
186     print("The Input is invalid!")
187 else:
188     print("Result:", result)
189 finally:
190     print("End of the process.")
```

شكل ۵-۹-مثالی از مدیریت خطاهای با ساختار **try/except/else/finally**

خروجی شکل ۵-۹:

اگر در ورودی مقدار Hello را قرار دهید، خروجی که می‌گیرید به این صورت خواهد بود:

---

*Enter a number: Hello*

*The Input is invalid!*

*End of the process.*

---

اگر در ورودی عدد صفر را قرار دهید، خروجی که می‌گیرید به این صورت خواهد بود:

---

*Enter a number: 0*

*Division by zero is not possible!*

*End of the process.*

---

اگر در ورودی عدد ۲۰ را قرار دهید، خروجی که می‌گیرید به این صورت خواهد بود:

---

*Enter a number: 20*

*Result: 0.5*

*End of the process.*

---

## ۵-۹) کاربرد مدیریت استثنایها

- افزایش پایداری برنامه: جلوگیری از متوقف شدن ناگهانی برنامه.
- ارائه پیامهای مناسب به کاربر: کاربران خطاهای فنی را درک نمی‌کنند؛ با مدیریت مناسب می‌توانید توضیحات قابل فهم ارائه دهید.
- سادهسازی رفع اشکال: شناسایی سریع منبع خطاهای.

# فصل دهم

مفهوم Scope در زبان برنامه‌نویسی پایتون

## ۱۰) مفهوم دامنه در زبان برنامه‌نویسی پایتون

دامنه<sup>۷۳</sup> در برنامه‌نویسی به محدوده‌ای اشاره دارد که در آن یک متغیر یا تابع تعریف شده قابل دسترسی است. در زبان پایتون، دامنه‌ها تعیین می‌کنند که کدام بخش از کد به متغیرها، توابع یا اشیاء دسترسی داشته باشد. در ک دقيق مفهوم دامنه برای نوشتن کدی ایمن و جلوگیری از مشکلاتی مانند تداخل نام‌ها ضروری است.

### ۱-۱) انواع Scope در پایتون

پایتون از یک سلسله مراتب مشخص برای جستجوی نام‌ها استفاده می‌کند که با قانون LEGB شناخته می‌شود. این قانون ترتیب جستجوی متغیرها را در چهار سطح دامنه مشخص می‌کند:

۱. دامنه محلی<sup>۷۴</sup>

۲. دامنه بسته<sup>۷۵</sup>

۳. دامنه سراسری<sup>۷۶</sup>

۴. دامنه داخلی<sup>۷۷</sup>

### ۱-۱-۱) دامنه محلی

متغیرهایی که در داخل یک تابع تعریف می‌شوند و فقط در همان تابع قابل دسترسی هستند.

<sup>73</sup> Scope

<sup>74</sup> Local Scope

<sup>75</sup> Enclosing

<sup>76</sup> Global Scope

<sup>77</sup> Built-in

### python example - Scope.py

```
1 # Scope in Python (دامنه دامنه در پایتون)
2
3 # Local Scope دامنه محلی
4 def my_function():
5     x = 45 # Local Scope
6     # در داخل تابع می‌توان ایکس را فراخوانی کرد
7     print(f"function Result is {x}")
8
9 my_function()
10 # اگر ایکس را در خارج از تابع فراخوانی کنیم، خطای خواهد داد
11 print(x)
```

شکل ۱-۱۰-۱- مثالی از دامنه محلی در پایتون

خروجی شکل ۱-۱۰:

---

*function Result is 45*

---

*NameError: name 'x' is not defined*

---

۱-۱-۲) دامنه بسته

متغیرهایی که در داخل یک تابع خارجی (بالا درست) تعریف شده‌اند و توسط توابع تو در تو قابل دسترسی هستند.

برای درک بهتر این دامنه به شکل ۱-۱-۲ توجه نمایید.

### python example - Scope.py

```
15 # Scope in Python (مبحث دامنه در پایتون)
16
17 # Enclosing دامنه بسته
18 def outer():
19     y = 20 # Enclosing Scope
20     def inner():
21         print(y) # دسترسی به متغیر Enclosing
22     inner()
23 outer()
```

شکل ۲-۱۰- مثالی از امنه بسته در پایتون

خروجی شکل ۲-۱۰:

---

20

---

### ۳-۱-۱۰) دامنه سراسری

متغیرهایی که در سطح بالاترین بخش کد تعریف می‌شوند و در کل برنامه قابل دسترسی هستند. به شکل ۳-۱۰ توجه نمایید.

### python example - Scope.py

```
27 # Scope in Python (مبحث دامنه در پایتون)  
28  
29 # Global Scope دامنه سراسری  
30 x = 30 # Global Scope  
31  
32 def my_function():  
33     print(f"function Result: {x}") # در داخل تابع می‌توان به ایکس دسترسی داشت  
34 my_function()  
35  
36 print(f"Global Scope: {x}") # در خارج از تابع می‌توان به ایکس دسترسی داشت
```

شکل ۱۰-۳- مثالی از دامنه سراسری در پایتون

خروجی شکل ۱۰-۳:

---

*function Result: 30*

*Global Scope: 30*

---

۱۰-۴) دامنه داخلی

تابع و نامهای از پیش تعریف شده در پایتون که به صورت پیش‌فرض در دسترس هستند، مانند `print()`, `len()` و غیره.

## ۱۰-۲) قانون جستجوی نامها یا قانون LEGB

وقتی پایتون با یک نام(نام متغیر یا تابع) مواجه می‌شود، آن را به ترتیب زیر جستجو می‌کند:

۱. ابتدا در دامنه محلی جستجو می‌کند.( $L = \text{Local}$ ).

۲. سپس در دامنه بسته جستجو می‌کند.( $E = \text{Enclosing}$ ).

۳. اگر پیدا نشد، به دامنه سراسری مراجعه می‌کند.( $G = \text{Global}$ ).

۴. در نهایت، در دامنه داخلی جستجو می‌شود.( $\text{Build-in}$ ).

اگر نام در هیچ‌یک از این دامنه‌ها یافت نشود، خطای `NameError` رخ می‌دهد.

### ۳-۱۰) متغیرهای سراسری و کلیدواژه `global`

در پایتون، متغیرهای درون یک تابع به‌طور پیش‌فرض محلی هستند. برای این‌که به این متغیرها را در خارج از تابع دسترسی داشته باشید باید متغیرهای مورد نظر را از حالت Local به Global تغییر دهید و برای این کار با توجه به شکل ۴-۱۰، باید از کلیدواژه `global` استفاده کنید.

python example - Scope.py

```
40 # Scope in Python (مبحث دامنه در پایتون)
41
42 # Global Scope and global keyword
43 # Convert local Scope to global scope
44
45 x = 10 # متغیر سراسری
46 def my_function():
47     global x
48     x = 20 # تغییر متغیر سراسری
49     print(f"In the Function: {x}")
50 my_function()
51 print(f"Out the Function: {x}") # مقدار تغییر کرده است
```

شکل ۴-۱۰-مثالی از تبدیل دامنه محلی به دامنه سراسری با کلیدواژه `global`

خروجی شکل ۴-۱۰:

*In the Function: 20*

*Out the Function: 20*

## ۴-۱۰) متغیرهای محلی و کلیدواژه nonlocal

کلیدواژه nonlocal در توابع داخلی استفاده می‌شود تا به متغیرهایی که در تابع بالادست تعریف شده‌اند دسترسی داشته و آن‌ها را تغییر دهد. برای درک بهتر کلیدواژه nonlocal به شکل ۵-۱۰ توجه نمایید.

python example - Scope.py

```
55 # Scope in Python (مبحث دامنه در پایتون)
56
57 # Local Scope and nonlocal keyword (دامنه محلی و استفاده از کلیدواژه nonlocal)
58 # Convert local Scope to global scope (تبديل دامنه محلی به دامنه سراسری)
59
60 def outer():
61     x = 10 # متغير Enclosing
62
63     def inner():
64         nonlocal x
65         x = 20 # تغيير مقدار متغير Enclosing
66         print("Inner:", x)
67
68     inner()
69     print("Outer:", x)
70
71 outer()
```

شکل ۵-۱۰-۱۰-۵-مثالی از تبدیل دامنه محلی به دامنه سراسری با کلیدواژه nonlocal

خروجی شکل ۵-۱۰:

---

*Inner: 20*

*Outer: 20*

---

نکات:

۱. متغیرهای محلی فقط در دامنه خودشان معتبر هستند.
۲. استفاده نادرست از `global` و `nonlocal` ممکن است کد را پیچیده کند و مانع از کدنویسی تمیز می‌شود.
۳. سعی کنید تا حد امکان از تعریف متغیرهای سراسری خودداری کرده و از متغیرهای محلی استفاده کنید.

# فصل یازدهم

آشنایی با برنامه‌نویسی شی‌گرا

## ۱۱) برنامه‌نویسی شی‌گرا

برنامه‌نویسی شی‌گرا<sup>۷۸</sup> یکی از روش‌های محبوب برای نوشتمن برنامه‌های است که به برنامه‌نویسان کمک می‌کند تا برنامه‌های بزرگ و پیچیده را به بخش‌های کوچک و قابل فهم تقسیم کنند. در این روش، از مفهومی به نام شی<sup>۷۹</sup> استفاده می‌شود. هر شی، یک موجودیت است که می‌تواند ویژگی‌ها و رفتارهای مخصوص خودش را داشته باشد. برای ساخت این اشیاء از کلاس<sup>۸۰</sup> استفاده می‌شود. کلاس‌ها مانند الگو یا نقشه‌ای هستند که از روی آن می‌توان اشیاء مختلفی ساخت.

در زبان برنامه‌نویسی پایتون، شی‌گرایی به راحتی قابل استفاده است و به برنامه‌نویسان کمک می‌کند کدهای منظم و بهتری بنویسند. با استفاده از شی‌گرایی می‌توان ویژگی‌ها<sup>۸۱</sup> و رفتارهای<sup>۸۲</sup> مربوط به یک موضوع خاص را در یک قالب مشخص تعریف کرد. مثلاً اگر در برنامه‌تان با موجودیت‌های دنیای واقعی مثل "ماشین" کار می‌کنید، می‌توانید یک کلاس برای ماشین تعریف کنید. این کلاس می‌تواند ویژگی‌هایی مثل رنگ و مدل ماشین و رفتارهایی مثل حرکت کردن و توقف کردن را داشته باشد. سپس از روی این کلاس می‌توان اشیاء مختلف (مانند ماشین‌های مختلف) ایجاد کرد.

از ویژگی‌های مهم برنامه‌نویسی شی‌گرا می‌توان به کپسوله‌سازی<sup>۸۳</sup>، وراثت<sup>۸۴</sup> و چندریختی<sup>۸۵</sup> اشاره کرد. این ویژگی‌ها کمک می‌کنند برنامه‌نویسان کدهای ساده همراه با قابلیت استفاده مجدد بنویسند.

کاربرد شی‌گرایی در پایتون بسیار گسترده است. این روش نه تنها برای برنامه‌های کوچک استفاده می‌شود، بلکه در پروژه‌های بزرگ مثل ساخت وب‌سایتها، برنامه‌های یادگیری ماشین و بازی‌ها هم به کار می‌رود. بسیاری از

<sup>78</sup> Object-Oriented Programming (OOP)

<sup>79</sup> Object

<sup>80</sup> Class

<sup>81</sup> Attributes

<sup>82</sup> Methods

<sup>83</sup> Encapsulation

<sup>84</sup> Inheritance

<sup>85</sup> Polymorphism

کتابخانه‌ها و فریمورک‌های معروف پایتون، مثل جنگو برای وب و تنسورفلو برای هوش مصنوعی، بر اساس اصول شی‌گرایی طراحی شده‌اند.

یکی از مزایای کلیدی برنامه‌نویسی شی‌گرا در پایتون، قابلیت مدل‌سازی مسائل پیچیده است. به عنوان مثال، در طراحی یک سیستم بانکی، می‌توان برای موجودیت‌های مختلف مانند حساب بانکی، مشتری و تراکنش، کلاس‌های جداگانه‌ای تعریف کرد که هر یک از این کلاس‌ها ویژگی‌ها و رفتارهای مخصوص به خود را دارند. از این طریق، هر بخش از سیستم به‌طور مستقل طراحی می‌شود و امكان تغییر یا گسترش هر بخش بدون اختلال در سایر بخش‌ها فراهم می‌گردد.

در نهایت، شی‌گرایی در پایتون نه تنها باعث کاهش پیچیدگی کد می‌شود، بلکه با سازمان‌دهی بهتر، خطاهای احتمالی را کاهش می‌دهد و به همکاری تیمی در پروژه‌های بزرگ کمک می‌کند. این رویکرد انعطاف‌پذیری بالایی را برای حل مسائل دنیای واقعی ارائه می‌دهد و یادگیری آن برای هر برنامه‌نویس پایتون امری ضروری و سودمند خواهد بود.

## ۱-۱۱) کلاس و شی در زبان برنامه‌نویسی پایتون

کلاس و شی از مفاهیم پایه‌ای در برنامه‌نویسی شی‌گرا هستند که در زبان پایتون نقش مهمی دارند. در این رویکرد، برنامه‌ها به کمک کلاس‌ها و اشیاء به بخش‌های کوچک‌تر و منظم‌تری تقسیم می‌شوند. این مفاهیم به ما اجازه می‌دهند تا مسائل دنیای واقعی را به راحتی مدل‌سازی کنیم.

### ۱-۱-۱۱) کلاس‌ها در پایتون

کلاس در پایتون مانند یک الگو یا قالب است که برای ایجاد اشیاء استفاده می‌شود. در واقع کلاس مجموعه‌ای از ویژگی‌ها و رفتارها است که می‌خواهیم برای یک موجودیت مشخص تعریف کنیم. به زبان ساده، کلاس مانند نقشه یا طرح اولیه‌ای برای ساخت اشیاء مختلف است. خود کلاس به تنها بی داده‌ای در اختیار ندارد، اما از آن می‌توان نمونه‌هایی به نام شیء ایجاد کرد.

### ۲-۱-۱۱) شیء در پایتون

شیء در پایتون یک نمونه<sup>۸۶</sup> از کلاس است. وقتی یک کلاس را تعریف می‌کنید، می‌توان از آن کلاس، اشیاء مختلفی ایجاد کرد که هر کدام ویژگی‌ها و رفتارهای مشخصی دارند. هر شیء در واقع یک موجودیت مستقل است که بر اساس کلاس ساخته می‌شود. به طور خلاصه، کلاس قالب است و شیء نمونه‌ای از آن قالب.

### ۳-۱-۱۱) ویژگی‌ها و رفتارهای کلاس و اشیاء

✓ ویژگی‌ها: ویژگی‌ها مشخصاتی هستند که برای هر شیء تعریف می‌شوند. این ویژگی‌ها معمولاً از طریق متغیرها بیان می‌شوند.

✓ رفتارها: رفتارها توابعی هستند که در کلاس تعریف می‌شوند و عملیاتی را روی ویژگی‌ها انجام می‌دهند. به این توابع در کلاس، متدها یا تابع وابسته گفته می‌شود.

<sup>۸۶</sup> Instance

#### ۱۱-۴) مثال‌هایی از کلاس و شیء

مثال ۱. فرض کنید می‌خواهید یک برنامه برای نمایش اطلاعات ماشین‌ها بنویسید. در اینجا می‌توانید یک کلاس به نام Car تعریف کنید و از آن چند شیء بسازید.

- کلاس: Car
- ویژگی‌ها: رنگ، مدل
- رفتارها: نمایش اطلاعات Car

مثال ۲. تصور کنید می‌خواهید کلاسی به نام Person تعریف کنید.

- کلاس: Person
- ویژگی‌ها: نام، سن
- رفتارها: نمایش اطلاعات Person

وقتی این کلاس را تعریف کنید، می‌توانید شیء‌هایی مانند شخص شماره ۱ و شخص شماره ۲ از آن ایجاد کنید. هر شیء دارای ویژگی‌های متفاوت (مثلًاً نام و سن) است، اما رفتارهای مشخص شده در کلاس را دارد.

#### ۱۱-۵) مزایای استفاده از کلاس و شیء در پایتون

۱. مدل‌سازی دنیای واقعی: با استفاده از کلاس‌ها و اشیاء، می‌توان موجودیت‌های واقعی را به راحتی در کد برنامه مدل‌سازی کرد.
۲. قابلیت استفاده مجدد از کد: یک کلاس می‌تواند بارها برای ایجاد اشیاء مختلف استفاده شود.

۳. سازمان دهی بهتر کد: کدها خواناتر و منظم‌تر می‌شوند و برنامه‌نویسان می‌توانند به راحتی آن را مدیریت کنند.

۴. انعطاف‌پذیری بالا: ویژگی‌ها و رفتارهای کلاس‌ها قابل تغییر و توسعه هستند.

#### ۱۱-۶) تعریف کلاس و شئ در پایتون

برای تعریف کلاس در پایتون، از کلیدواژه‌ی `class` شروع کرده و یک نام دلخواه برای کلاس داده و سپس دو نقطه روی هم و در خط بعدی با اعمال دندانه‌گذاری، بدن کلاس را کامل می‌کنید.

---

*class MyClass:*

*x = 5*

---

برای تعریف شئ از یک کلاس، ابتدا یک متغیر ایجاد کرده و سپس مقدار متغیر را کلاس ساخته شده، دهید. برای اجرای ویژگی‌ها و رفتارها طبق کد زیر عمل کنید.

---

*p1 = MyClass()*

*print(p1.x)* # برای فراخوانی ویژگی که در داخل کلاس می‌باشد

---

### ۷-۱-۱۱) تابع `__init__()`

تابع `__init__()` در زبان برنامه‌نویسی پایتون یکی از توابع وابسته ویژه<sup>۸۷</sup> است که به عنوان سازنده<sup>۸۸</sup> شناخته می‌شود. این تابع به طور خودکار هنگام ایجاد یک شیء از روی کلاس فراخوانی می‌شود و وظیفه آن مقداردهی اولیه ویژگی‌های شیء است. به عبارت ساده‌تر، تابع `__init__()` کمک می‌کند تا مقدار اولیه را به ویژگی‌های شیء نسبت دهد.

### ۱-۷-۱-۱۱) نقش تابع `__init__()`

هنگامی که از یک کلاس شیء ایجاد می‌کنید، ممکن است بخواهید ویژگی‌های خاصی مانند نام، سن، رنگ و غیره را به آن شیء نسبت دهید. تابع `__init__()` برای این منظور طراحی شده است و به طور خودکار توسط پایتون هنگام ساخت شیء فراخوانی می‌شود.

این تابع معمولاً شامل آرگومان `self` می‌باشد که به شیء جاری اشاره می‌کند و برای دسترسی به ویژگی‌های کلاس استفاده می‌شود. می‌توانید علاوه بر `self`، آرگومان‌های دیگری نیز به این تابع اضافه کنید تا مقدار اولیه را دریافت و ذخیره کنید.

### ۲-۷-۱-۱۱) ساختار تابع `__init__()`

ساختار کلی این تابع همانند شکل ۱-۱۱ است:

---

<sup>۸۷</sup> Special Methods

<sup>۸۸</sup> Constructor

### python example - OOP.py

```
1 # Object-Oriented Programming (OOP) in Python
2
3 # __init__ Synatx:
4
5 """
6 class ClassName:
7     def __init__(self, arg1, arg2, ...):
8         self.attribute1 = arg1
9         self.attribute2 = arg2
10    """
```

شکل ۱۱-۱- ساختار کلی تابع `__init__()` در داخل کلاس‌های پایتون

توضیحات شکل ۱۱-۱:

- `Self`: اشاره به شیءی که در حال ساخته شدن است.

- `attribute1, attribute2`: ویژگی‌هایی هستند که برای شیء تعریف می‌کنید و مقدارشان از

- آرگومان‌های ورودی (`arg1, arg2, ...`) گرفته می‌شود.

### ۳-۷-۱-۱۱) مثالی از `__init__()`

در شکل ۲-۱۱ کلاسی به نام `Person` تعریف شده و از `__init__()` برای مقداردهی اولیه ویژگی‌های شیء

استفاده می‌شود:

### python example - OOP.py

```
14 # Object-Oriented Programming (OOP) in Python
15
16 # __init__ Example:
17
18 class Person:
19     def __init__(self, name, age):
20         self.name = name # ویژگی name
21         self.age = age   # ویژگی age
22
23     ایجاد اشیاء از کلاس Person
24 person1 = Person("Ali", 25)
25 person2 = Person("Sara", 30)
26
27 نمایش ویژگی‌های اشیاء #
28 print(person1.name, person1.age)
29 print(person2.name, person2.age)
```

شکل ۱۱-۲- مثالی از تابع `__init__` در درون کلاس‌های پایتون

خروجی شکل ۱۱-۲:

---

*Ali 25*

---

*Sara 30*

---

توضیحات شکل ۱۱-۲:

• هنگام ساخت `person1` و `person2`، تابع `__init__` به طور خودکار فراخوانی می‌شود و مقادیر

و `name` و `age` را به ویژگی‌های شیء نسبت می‌دهد.

• ویژگی‌ها با استفاده از `self` به هر شیء نسبت داده می‌شوند.

## ۱۱-۷-۴) ویژگی‌های تابع `__init__()`

۱. فراخوانی خودکار: نیازی به فراخوانی مستقیم این تابع نداریم. این تابع به طور خودکار هنگام ساخت شیء اجرا می‌شود.

۲. مقداردهی اولیه: از این تابع برای مقداردهی اولیه ویژگی‌های شیء استفاده می‌شود.

۳. سفارشی‌سازی رفتار کلاس: می‌توان ویژگی‌های کلاس را متناسب با نیاز خدمان در زمان ایجاد شیء تنظیم کرد.

## ۱۱-۷-۵) مفهوم `self`

در برنامه‌نویسی شیء‌گرا پایتون، "self" یک کلمه کلیدی خاص است که نمایانگر نمونه‌ای از کلاس است. این کلمه در توابع وابسته نمونه برای دسترسی به ویژگی‌های نمونه و سایر توابع وابسته کلاس استفاده می‌شود. به عبارت ساده‌تر، وقتی شما یک کلاسی ایجاد می‌کنید، برای این‌که هر شیء، یک نمونه منحصر به‌فرد باشد، به `self` نیاز است. برای درک بهتر، به شکل ۱۱-۳ توجه نمایید.

### نکاتی درباره "self"

- ✓ هنگامی که شما یک شیء از یک کلاس ایجاد می‌کنید، "self" نماینده آن شیء خاص است. این اجازه می‌دهد که توابع وابسته روی داده‌ها (ویژگی‌ها) ذخیره شده در آن نمونه عمل کند.
- ✓ پایتون نیاز دارد تا شما آن را به عنوان اولین پارامتر تابع وابسته نمونه به صراحة اعلام کنید.
- ✓ در حالی که "self" یک کلمه کلیدی رزرو شده نیست، از نظر فنی شما می‌توانید آن را هر نامی بگذارید، اما انحراف از "self" می‌تواند خوانایی کد را برای سایر توسعه‌دهندگان پایتون کاهش دهد.

```

33 # Object-Oriented Programming (OOP) in Python برنامه نویسی شئ گرایی در پایتون
34
35 # __init__ Example:
36 class Person:
37     def __init__(self, name, age):
38         self.name = name # ویژگی نمونه
39         self.age = age   # ویژگی نمونه
40
41     def greet(self):
42         return f'Hello, my name is {self.name} and I am {self.age} years old.'
43
44 ساخت یک شئ از کلاس (ایجاد یه نمونه از کلاس)
45 Reza = Person("REZA", 24)
46 Mahdi = Person("MAHDI", 30)
47
48
49 فراخوانی رفتار برای نمونه ایجاد شده
50 print(Reza.greet())
51 print(Mahdi.greet())

```

شکل ۱۱-۳- مثالی برای درک بهتر *self*

خروجی شکل ۱۱-۳:

*Hello, my name is REZA and I am 24 years old.**Hello, my name is MAHDI and I am 30 years old.*

---

### ۱۱-۷-۶) نحوه عملکرد *self*

➤ تابع *init* نمونه را مقداردهی اولیه می‌کند. "self" تضمین می‌کند که ویژگی‌های *name* و *age* به

نمونه اختصاص می‌یابد.

► در داخل هر تابع وابسته‌ی نمونه، شما می‌توانید با استفاده از "self.attribute\_name" به ویژگی‌ها

دسترسی پیدا کنید.

► فراخوانی سایر توابع وابسته: در داخل یک کلاس، توابع وابسته می‌توانند با استفاده از

"self.method\_name()" روی همان نمونه فراخوانی شوند.

نکته: بدون self، دسترسی به نمونه را از دست می‌دهید.

با توجه به شکل ۱۱-۴، اگر self را در تعاریف توابع وابسته حذف کنید، پایتون خطا خواهد داد.

python example - OOP.py

```
55 # Object-Oriented Programming (OOP) in Python
56
57 # __init__ Example:
58 class Person:
59     def __init__(name, age): # Missing 'self'
60         self.name = name    # Error: 'self' is not defined
61         self.age = age
62     def greet():
63         return f"Hello, my name is {self.name} and I am {self.age} years old."
64
65 Reza = Person("Reza", 24)
66 print(Reza.greet())
```

شکل ۱۱-۴-مثالی از حذف self در تعریف توابع وابسته

خروجی شکل ۱۱-۴:

---

*Traceback (most recent call last):*

*Reza = Person("Reza", 24)*

*TypeError: Person.\_\_init\_\_() takes 2 positional arguments but 3 were given*

---

نکات:

- ✓ همیشه "self" را به عنوان اولین پارامتر در هنگام تعریف توابع وابسته‌ی نمونه قرار دهید.
- ✓ از "self" برای تمایز بین ویژگی‌های نمونه و متغیرهای محلی یا سطح کلاس استفاده کنید.
- ✓ این امکان را فراهم می‌کند که هر نمونه حالت خاص خود را حفظ کند و رفتار پویایی داشته باشد.

#### ۱۱-۸) تابع `__str__()`

تابع `\_\_str\_\_()` یکی از توابع وابسته‌ی جادویی<sup>۸۹</sup> در پایتون است که برای تعریف نحوه نمایش "خوانای کاربر"<sup>۹۰</sup> از یک شیء استفاده می‌شود. وقتی از تابع `str()` روی یک شیء استفاده کنید یا شیء را مستقیماً با توابعی مثل `print()` نمایش دهید، این تابع وابسته مانند شکل ۱۱-۵ فراخوانی می‌شود. این تابع معمولاً برای بازگشت یک رشته خوانا و توصیفی از شیء استفاده می‌شود که می‌تواند برای کاربران مناسب باشد.

---

<sup>۸۹</sup> Magic Methods

<sup>۹۰</sup> user-friendly representation

### python example - OOP.py

```
70 # Object-Oriented Programming (OOP) in Python
71
72 # __str__ Example:
73
74 class Person:
75     def __init__(self, name, age):
76         self.name = name
77         self.age = age
78     def __str__(self):
79         return f'Name: {self.name}, Age: {self.age}'
80
81 # ساخت یک شیء
82 person = Person("Ali", 25)
83
84 # استفاده از print یا str
85 print(person)
86 print(str(person))
```

شکل ۱۱-۵-مثالی از تابع `__str__` در کلاس‌های پایتون

---

خروجی شکل ۱۱-۵:

*Name: Ali, Age: 25*

*Name: Ali, Age: 25*

---

### ۱۱-۶-تابع `__repr__()`

تابع `__repr__()` یکی دیگر از توابع وابسته‌ی جادویی در پایتون است که وظیفه ارائه نمایشی رسمی از شیء را بر عهده دارد. این تابع وابسته، برای توسعه‌دهندگان طراحی شده و هدف آن، این است که اگر رشته‌ای در کدها

استفاده شده باشد، آن رشته را بازمی‌گرداند. در شکل ۱۱-۶، به این موضوع در ادامه به کاربرد تابع `__repr__()` پرداخته می‌شود.

- ✓ نمایش رسمی<sup>۹۱</sup>: وقتی از تابع `repr()` روی یک شیء استفاده می‌کنید یا در محیط‌هایی مثل ترمینال پایتون، شیء را تایپ می‌کنید، این تابع فراخوانی می‌شود.
- ✓ بازسازی شیء: توصیه می‌شود رشته بازگشتی از این تابع وابسته به گونه‌ای باشد که با اجرای آن، بتوان همان شیء را بازسازی کرد (هرچند این یک قانون اجباری نیست).

#### python example - OOP.py

```
برنامه نویسی شع گرایی در پایتون # Object-Oriented Programming (OOP) in Python
90  # Object-Oriented Programming (OOP) in Python
91
92  # __repr__ Example:
93  class Person:
94      def __init__(self, name, age):
95          self.name = name
96          self.age = age
97
98      def __repr__(self):
99          return f"Person(name={self.name!r}, age={self.age!r})"
100
101
102  # ساخت یک شیء
103 person = Person("Ali", 25)
104
105  # استفاده از repr
106 print(repr(person))
```

شكل ۱۱-۶- مثالی از تابع `__repr__()` در کلاس‌های پایتون

خروجی شکل ۱۱-۶:

---

<sup>۹۱</sup> Developer-Friendly

---

*Person(name='Ali', age=25)*

---

استفاده بدون تعریف `__str__()`: اگر تابع وابسته `__str__()` را تعریف نکنید، پایتون به طور پیشفرض از تابع وابسته `__repr__()` استفاده می‌کند. اگر هیچ‌کدام تعریف نشده باشند همانند شکل ۷-۱۱، شیء به صورت زیر نمایش داده می‌شود:

---

`'<ClassName object at memory_address>'`

---

python example - OOP.py

```
برنامه نویسی شیء گرایی در پایتون # Object-Oriented Programming (OOP) in Python
110
111
112 ساخت کلاس بدون __str__ ، __repr__
113 class Person:
114     def __init__(self, name, age):
115         self.name = name
116         self.age = age
117
118 person = Person("Ali", 25)
119
120 print(person)
```

شكل ۷-۱۱-۱- مثالی از ساخت کلاس بدون `__str__` و `__repr__`

خروجی شکل ۷-۱۱:

---

*<\_\_main\_\_.Person object at 0x0000026597646A50>*

---

## ۱۰-۱-۱۱) تفاوت بین `__str__()` و `__repr__()`

تابع `__str__()` : نمایش رشته‌ای، مناسب برای کاربران. باید "خوانا و دوستانه" باشد.

تابع `__repr__()` : نمایش رشته‌ای، مناسب برای توسعه‌دهندگان. باید شامل اطلاعات کافی باشد تا بتوان با آن، شیء را بازسازی کرد.

برای درک بهتر این تفاوت به شکل ۸-۱۱ توجه نمایید.

python example - OOP.py

```
124 # Object-Oriented Programming (OOP) in Python
125
126 # The difference between __str__ vs __repr__
127 class Person:
128     def __init__(self, name, age):
129         self.name = name
130         self.age = age
131     def __str__(self):
132         return f"Name: {self.name}, Age: {self.age}"
133     def __repr__(self):
134         return f"Person(name={self.name!r}, age={self.age!r})"
135
136
137 person = Person("Ali", 25)
138 print(person)
139 print(str(person))
140 print(repr(person))
```

شکل ۸-۱۱-۱) تفاوت بین `__str__()` و `__repr__()`

خروجی شکل ۸-۱۱:

---

Name: Ali, Age: 25

Name: Ali, Age: 25

*Person(name='Ali', age=25)*

---

نکات:

- محتوای رشته بازگشتی از `\_\_str\_\_()` باید برای کاربران معمولی خوانا و معنادار باشد.
- تابع `\_\_str\_\_()` هنگام استفاده از `print().\_\_str\_\_()` یا `str().\_\_str\_\_()` روی شیء فراخوانی می‌شود.
- استفاده از `\_\_str\_\_()` به شما امکان می‌دهد نمایش بهتری برای اشیاء کلاس خود در خروجی ایجاد کنید که کاربرپسند باشد.
- انتخاب مناسب بین `\_\_str\_\_()` و `\_\_repr\_\_()` : اگر فقط یکی از این دو را تعریف کنید، معمولاً `\_\_repr\_\_()` اولویت دارد.

### ۱۱-۱-۱۱) توابع وابسته در کلاس‌های پایتون

در پایتون، توابع وابسته توابعی هستند که درون یک کلاس تعریف می‌شوند و معمولاً برای انجام عملیاتی روی داده‌های یک شیء یا دسترسی به آن‌ها استفاده می‌شوند. توابع وابسته با استفاده از کلیدواژه `def` تعریف می‌شوند.

#### نواع توابع وابسته در کلاس‌های پایتون

##### ۱. توابع وابسته‌ی نمونه<sup>۹۲</sup>

- به داده‌ها و ویژگی‌های یک نمونه خاص از کلاس دسترسی دارند.
- اولین آرگومان آن‌ها باید `self` باشد که به نمونه جاری اشاره می‌کند.

---

<sup>۹۲</sup> Instance Methods

## ۲. توابع وابسته‌ی کلاس<sup>۹۳</sup>

- به کل کلاس (و نه فقط یک نمونه خاص) دسترسی دارند.
- با دکوریتور `@classmethod` تعریف می‌شوند.
- اولین آرگومان آن‌ها `cls` است که به خود کلاس اشاره می‌کند.

## ۳. توابع وابسته‌ی ایستا<sup>۹۴</sup>

- به کلاس یا نمونه دسترسی مستقیم ندارند.
- با دکوریتور `@staticmethod` تعریف می‌شوند.
- برای انجام عملیاتی که وابسته به داده‌های کلاس یا نمونه نیستند استفاده می‌شوند.

## ۱۱-۱-۱۱-۲) تعریف و مثال برای هر نوع تابع وابسته

### ۱. تابع وابسته نمونه

---

<sup>۹۳</sup> Class Methods

<sup>۹۴</sup> Static Methods

### python example - OOP.py

```
144 # Object-Oriented Programming (OOP) in Python
145
146 # Instance Methods Example: مثال توابع وابسته نمونه
147
148 class Person:
149     def __init__(self, name, age):
150         self.name = name # ویژگی نمونه
151         self.age = age
152     def greet(self):
153         return f"Hello, my name is {self.name} and I am {self.age} years old."
154
155 # استفاده از تابع وابسته نمونه
156 person1 = Person("Ali", 25)
157 print(person1.greet())
```

شکل ۱۱-۹- مثالی برای تابع وابسته نمونه

خروجی شکل ۱۱-۹:

---

*Hello, my name is Ali and I am 25 years old.*

---

۲. تابع وابسته کلاس

### python example - OOP.py

```
163 # Class Methods Example:  
164 class Person:  
165     population = 0 # ويژگی کلاس  
166     def __init__(self, name):  
167         self.name = name  
168         Person.population += 1  
169     @classmethod  
170     def get_population(cls):  
171         return f"There are {cls.population} people."  
172  
173 # استفاده از تابع وابسته کلاس  
174 print(Person.get_population())  
175  
176 person1 = Person("Ali")  
177 print(Person.get_population())  
178  
179 person2 = Person("Reza")  
180 print(Person.get_population())
```

شكل ۱۰-۱۱-۱۱- مثالی برای تابع وابسته کلاس

خروجی شکل ۱۰-۱۱:

---

*There are 0 people.*

*There are 1 people.*

*There are 2 people.*

---

۳. تابع وابسته ایستا

## python example - OOP.py

```
184 # Object-Oriented Programming (OOP) in Python
185
186 # Static Methods Example: مثال توابع وابسته ایستا
187 class MathUtils:
188     @staticmethod
189     def add(a, b):
190         return a + b
191
192 # استفاده از تابع وابسته ایستا
193 print(MathUtils.add(5, 3))
194
195 print(MathUtils.add(9, 1))
196
197 print(MathUtils.add(20, 4))
```

شكل ۱۱-۱۱-۱۱- مثالی برای تابع وابسته ایستا

خروجی شکل ۱۱-۱۱:

---

۸

۱۰

۲۴

---

نکات:

۱. تابع وابسته‌ی نمونه:

- برای کار با داده‌های مرتبط با یک نمونه خاص از کلاس استفاده می‌شود.

- نیاز به self دارند.

## ۲. تابع وابسته‌ی کلاس:

○ می‌توانند به داده‌های سطح کلاس دسترسی داشته باشند.

○ نیازی به یک نمونه خاص ندارند.

## ۳. تابع وابسته‌ی ایستا:

○ به هیچ داده‌ای از کلاس یا نمونه دسترسی ندارند.

○ برای عملیات‌هایی مناسب هستند که به کلاس یا نمونه وابسته نیستند.

python example - OOP.py

```
201 # Object-Oriented Programming (OOP) in Python برنامه نویسی شئ‌گرایی در پایتون
202
203 # Methods Example: مثال استفاده از توابع وابسته مختلف در یک مسئله
204 class Calculator:
205     def __init__(self, name):
206         self.name = name
207
208     def instance_method(self, x, y):
209         return f'{self.name} adds: {x + y}'
210
211     @classmethod
212     def class_method(cls, x, y):
213         return f'Class adds: {x + y}'
214
215     @staticmethod
216     def static_method(x, y):
217         return f'Static adds: {x + y}'
218
219 # استفاده از توابع وابسته
220 calc = Calculator("MyCalculator")
221 print(calc.instance_method(3, 4)) # تابع وابسته نمونه
222 print(Calculator.class_method(3, 4)) # تابع وابسته کلاس
223 print(Calculator.static_method(3, 4)) # تابع وابسته ایستا
```

شکل ۱۱-۱۲- تعریف چند تابع وابسته‌ی مختلف در یک کلاس

*MyCalculator adds: 7*

*Class adds: 7*

*Static adds: 7*

نکات:

- از تابع وابسته نمونه برای دسترسی و تغییر ویژگی‌های یک شیء استفاده می‌شود.
- از تابع وابسته کلاس برای دسترسی و تغییر ویژگی‌های کلاس استفاده می‌شود.
- از تابع وابسته ایستا برای عملیات‌های مستقل که به داده‌های کلاس یا نمونه نیاز ندارند، استفاده می‌شود.

## ۱۱-۲) وراثت در پایتون

وراثت در پایتون یکی از مفاهیم اصلی برنامه‌نویسی شیء‌گرا است که به یک کلاس (کلاس فرزند) اجازه می‌دهد تا ویژگی‌ها و توابع وابسته یک کلاس دیگر (کلاس والد) را به ارث ببرد. این امر باعث افزایش استفاده مجدد از کد و سازماندهی منطقی می‌شود.

### ۱۱-۲-۱) ویژگی‌های کلیدی وراثت

- ❖ کلاس والد (کلاس پایه): کلاسی که از آن ارث بری می‌شود.
- ❖ کلاس فرزند (کلاس مشتق شده): کلاسی که از کلاس والد ارث بری می‌کند.

- ❖ استفاده مجدد از کد: کلاس‌های فرزند می‌توانند از توابع وابسته و ویژگی‌های کلاس‌های والد استفاده کنند.
- ❖ بازنویسی توابع وابسته: کلاس‌های فرزند می‌توانند توابع وابسته کلاس والد را بازنویسی کنند.
- ❖ وراثت چندگانه: یک کلاس فرزند می‌تواند از چندین کلاس والد ارث بری کند.

## ۱۱-۲-۲) نحوه استفاده از وراثت در پایتون

python example - OOP.py

```

227 # Object-Oriented Programming (OOP) in Python
228
229 # Inheritance in Python
230 class ParentClass:
231     # Parent class definition
232     pass
233
234 class ChildClass(ParentClass):
235     # Child class definition
236     pass
237

```

شکل ۱۱-۱۳- مثالی از مفهوم وراثت در پایتون

## ۱۱-۲-۳) مثالی از وراثت تک‌گانه

python example - OOP.py

```
240 # Object-Oriented Programming (OOP) in Python  
241  
242 # Inheritance in Python  
243 # وراثت در پایتون  
244 # تک گانه  
245 class Animal:  
246     def speak(self):  
247         return "I make a sound."  
248  
249 class Dog(Animal):  
250     def speak(self):  
251         return "Woof! Woof!"  
252  
253 animal = Animal()  
254 dog = Dog()  
255  
256 print(animal.speak())  
257 print(dog.speak())
```

شکل ۱۱-۱۴- مثالی از وراثت تک‌گانه در پایتون

خروجی شکل ۱۱-۱۴:

---

*I make a sound.*

*Woof! Woof!*

---

نکته: کلاس‌های فرزند می‌توانند توابع وابسته‌ی کلاس والد را بازنویسی<sup>۹۵</sup> کنند.

---

<sup>۹۵</sup> Method Overriding

### python example - OOP.py

```
261 # Object-Oriented Programming (OOP) in Python
262
263 # Inheritance in Python
264
265 # overriding کلاس والد توسط کلاس های فرزند
266 class Parent:
267     def greet(self):
268         return "Hello from the parent class!"
269
270 class Child(Parent):
271     def greet(self):
272         return "Hello from the child class!"
273
274 # Test overriding
275 obj = Child()
276 print(obj.greet())
```

شکل ۱۱-۱۵- مثالی از بازنویسی کلاس والد توسط کلاس های فرزند

خروجی شکل ۱۱-۱۵:

---

*Hello from the child class!*

---

### ۱۱-۲-۴) تابع وابسته‌ی **super()**

تابع وابسته‌ی **super()** در پایتون یکی از مفاهیم کلیدی در برنامه‌نویسی شیءگرا است که برای دسترسی به توابع وابسته و ویژگی‌های کلاس پایه (والد) از درون یک کلاس مشتق (فرزند) مورد استفاده قرار می‌گیرد. این تابع امکان فراخوانی توابع وابسته کلاس والد را بدون نیاز به ارجاع مستقیم به نام آن کلاس فراهم می‌کند و به طور خاص برای موقعی کاربرد دارد که توابع وابسته والد باز تعریف شده‌اند.

یکی از اهداف اصلی استفاده از `super()`، ساده کردن مدیریت ارث بری در کلاس‌ها، خصوصاً در ساختارهای پیچیده مانند ارث بری چندگانه و چندسطحی است. این تابع از مکانیزم ترتیب جستجوی تابع وابسته<sup>۹۶</sup> استفاده می‌کند تا تابع وابسته یا ویژگی موردنظر را در سلسله‌مراتب کلاس‌ها پیدا کند. این ترتیب مشخص می‌کند که پایتون به چه صورت و با چه اولویتی کلاس‌ها را در هنگام جستجوی توابع وابسته پیمایش می‌کند. با این روش، `super()` از مشکلات رایج در ارث بری چندگانه، مانند دوبار اجرای توابع وابسته والد، جلوگیری می‌کند.

تابع `super()` همچنین در ساختارهای کلاس‌ها بسیار مفید است. به کمک آن، کلاس فرزند می‌تواند ویژگی‌های پایه کلاس والد را مقداردهی کرده و سپس تغییرات یا ویژگی‌های خاص خود را اضافه کند. این ویژگی به ویژه در موقعی که ساختار کلاس پیچیده است و مدیریت دستی ارث بری می‌تواند منجر به خطاهای منطقی شود، اهمیت دارد. به این ترتیب، `super()` نه تنها کدنویسی را کوتاهتر و خواناتر می‌کند، بلکه با مدیریت بهتر سلسله‌مراتب کلاس‌ها، به توسعه‌دهندگان امکان می‌دهد از کد خود مجدد استفاده کرده و از تکرار کد جلوگیری کنند. به شکل ۱۱-۱۶ توجه نمایید.

<sup>۹۶</sup> MRO - Method Resolution Order

python example - OOP.py

```
280 # Object-Oriented Programming (OOP) in Python  
281  
282 # Inheritance in Python  
283 # مثالی از تابع Super()  
284  
285 class Parent:  
286     def greet(self):  
287         return "Hello from Parent!"  
288  
289 class Child(Parent):  
290     def greet(self):  
291         return super().greet() + " And hello from Child!"  
292  
293 # Test super()  
294 obj = Child()  
295 print(obj.greet())
```

شکل ۱۱-۱۶-مثال مربوط به تابع *super* در پایتون

خروجی شکل ۱۱-۱۶:

---

*Hello from Parent! And hello from Child!*

---

### ۱۱-۲-۵) وراثت چندگانه در پایتون

پایتون از وراثت چندگانه پشتیبانی می‌کند و به یک کلاس اجازه می‌دهد از چندین کلاس والد ارث بری کند مانند

شکل ۱۱-۱۷.

### python example - OOP.py

```
299 # Object-Oriented Programming (OOP) in Python
300
301 # Inheritance in Python
302 # وراثت در پایتون
303 # وراثت چندگانه در پایتون
304 class A:
305     def method_a(self):
306         return "Method from class A"
307
308 class B:
309     def method_b(self):
310         return "Method from class B"
311
312 class C(A, B):
313     pass
314
315 # Test multiple inheritance
316 obj = C()
317 print(obj.method_a())
318 print(obj.method_b())
```

شکل ۱۷-۱۱-مثالی از وراثت چندگانه در پایتون

خروجی شکل ۱۷-۱۱:

---

*Method from class A*

*Method from class B*

---

## ۱۱-۲-۶) وراثت چندسطحی در پایتون

وراثت چندسطحی در پایتون به حالتی گفته می‌شود که یک کلاس از کلاس دیگری ارث بری کند و سپس یک کلاس سوم از کلاس دوم ارث بری کند، و این زنجیره می‌تواند ادامه یابد. این نوع وراثت امکان انتقال ویژگی‌ها و رفتارها از یک کلاس پایه به چندین سطح از کلاس‌های مشتق شده را فراهم می‌کند.

python example - OOP.py

```
برنامه نویسی شی گرایی در پایتون # Object-Oriented Programming (OOP) in Python
321 # Inheritance in Python
322 # وراثت در پایتون
323 # multi-level inheritance
324 # وراثت چندسطحی
325
326 class Animal:
327     def breathe(self):
328         return "I am breathing"
329
330 class Mammal(Animal):
331     def walk(self):
332         return "I am walking"
333
334 class Dog(Mammal):
335     def bark(self):
336         return "Woof! Woof!"
337
338 # Test multi-level inheritance
339 dog = Dog()
340 print(dog.breathe())
341 print(dog.walk())
342 print(dog.bark())
```

شکل ۱۱-۱۸-۱۸-مثالی از وراثت چندسطحی در پایتون

خروجی شکل ۱۱-۱۸:

*I am breathing*

*I am walking*

## *Woof! Woof!*

---

### ۱۱-۳) چندريختی در پايتون

چندريختی در پايتون يك مفهوم برنامه‌نويسی شيء‌گرا است که اجازه مى‌دهد از يك رابط يکسان (تابع وابسته) برای اشیاء از انواع مختلف استفاده شود. اين امر رویکردي يکپارچه و انعطاف‌پذير برای نوشتن کد را ممکن می‌سازد، زира می‌توانيد از يك تابع يا تابع وابسته يکسان در كلاس‌های مختلف استفاده کنید، به شرطی که همان رابط را پياده‌سازی کنند.

چندريختی در برنامه‌نويسی شيء‌گرا به اين مفهوم اشاره دارد که يك تابع وابسته، يا شيء می‌تواند رفتارهای مختلفی داشته باشد، بسته به نوع داده‌ها يا كلاس‌هایی که با آن‌ها کار می‌کند. در پايتون، اين ويزگی به دليل پويایي و تايپ‌گذاري ضعيف زبان بسيار پرنگ است و انعطاف زيادي در كدنويسی فراهم می‌کند.

چندريختی به ما اجازه مى‌دهد که از يك رابط يا يك تابع وابسته مشترك برای اشیای مختلف استفاده کنیم، بدون اينکه نگران نوع يا كلاس آن اشیا باشيم. به عبارت ساده‌تر، اگر دو يا چند كلاس توابع وابسته با نام يکسان داشته باشند، می‌توان با استفاده از همان تابع وابسته، رفتارهای متفاوتی در اشیای مختلف مشاهده کرد. اين ويزگی باعث می‌شود کد ساده‌تر و انعطاف‌پذيرتر باشد.

### ۱۱-۳) انواع چندريختی در پايتون

۱. چندريختی مبتنی بر ارثبری: در اين حالت، كلاس‌های فرزند توابع وابسته كلاس والد را بازنويسی می‌کنند. بنابراین، تابع وابسته مشترك در كلاس‌های مختلف رفتار متفاوتی دارد.

۲. چندریختی مبتنی بر Duck Typing: در این رویکرد، نوع یا کلاس شیء اهمیت ندارد؛ بلکه رفتار یا توابع وابسته‌ای که آن شیء پشتیبانی می‌کند مهم است. این ویژگی در پایتون که زبان دینامیک است، بسیار پرکاربرد است.

### ۲-۳-۱۱) ویژگی‌های چندریختی

• یک زیر کلاس می‌تواند پیاده‌سازی خاص خود را برای یک تابع وابسته تعریف شده در کلاس والد ارائه دهد.

• پایتون اجازه می‌دهد تا یک تابع واحد روی انواع مختلف داده عمل کند.

• چندریختی پایتون بر اساس فلسفه "اگر مانند اردک به نظر می‌رسد و مانند اردک قورقور می‌کند، پس اردک است" استوار است. اشیاء بر اساس رفتار خود و نه نوع واقعی آن‌ها مورد ارزیابی قرار می‌گیرند.

### ۳-۳-۱۱) ترکیب چندریختی با ارثبری و Duck Typing

#### ۱-۳-۳-۱۱) چندریختی با ارثبری

در این حالت، یک تابع وابسته مشترک در کلاس والد در کلاس‌های فرزند رفتار متفاوتی خواهد داشت. این حالت، یکی از اصول چندریختی است. برای درک موضوع به شکل ۱۹-۱۱ توجه نمایید.

python example - OOP.py

```
346 # Object-Oriented Programming (OOP) in Python
347
348 # Polymorphism in Python
349
350 # ترکیب چندریختی با ارث بری
351 class Animal:
352     def sound(self):
353         return "Some generic sound"
354 class Dog(Animal):
355     def sound(self):
356         return "Woof! Woof!"
357 class Cat(Animal):
358     def sound(self):
359         return "Meow!"
360
361 animals = [Dog(), Cat(), Animal()]
362 for animal in animals:
363     print(animal.sound())
```

شکل ۱۹-۱۱- مثالی از چندریختی با ارث بری

خروجی شکل ۱۹-۱۱:

---

*Woof! Woof!*

*Meow!*

*Some generic soun*

---

## ۱۱-۳-۲) چند ریختی با Duck Typing

در پایتون، نیازی نیست که کلاس‌ها حتماً از یک والد مشترک ارثبری کنند. اگر تابع وابسته یا ویژگی موردنیاز در کلاس تعریف شده باشد، پایتون آن را به درستی اجرا می‌کند.

python example - OOP.py

```
367 # Object-Oriented Programming (OOP) in Python  
368  
369 # Polymorphism in Python  
370  
371 # Duck Typing چند ریختی با  
372 class Bird:  
373     def fly(self):  
374         return "Flying high"  
375 class Airplane:  
376     def fly(self):  
377         return "Taking off"  
378     def lift_off(entity):  
379         print(entity.fly())  
380  
381 lift_off(Bird())  
382 lift_off(Airplane())
```

شکل ۱۱-۲۰- مثالی از ترکیب چند ریختی با *Duck Typing*

خروجی شکل ۱۱-۲۰:

---

*Flying high*

*Taking off*

---

نکته: اگرچه چندريختی اغلب با ارثبری همراه است، اما اين دو مفهوم يكسان نيستند. چندريختی می‌تواند بدون ارثبری نيز کار کند (مثل Duck Typing). اما ارثبری به عنوان يكى از روش‌های رايچ برای پياده‌سازی چندريختی عمل می‌کند.

#### ۱۱-۳-۴) مزایای چندريختی

- ✓ کاهش وابستگی: کدها انعطاف‌پذیرتر و مستقل از نوع اشیا نوشته می‌شوند.
- ✓ کدخوانی بهتر: کد مختصرتر و ساده‌تر می‌شود.
- ✓ افزایش قابلیت استفاده مجدد: یک رابط مشترک برای انواع مختلف اشیا فراهم می‌شود.

# فصل دوازدهم

ماژول‌ها در پایتون

## ۱۲) ماثول‌ها در پایتون

ماژول‌ها در پایتون فایل‌هایی هستند که شامل کد پایتون (مثل توابع، کلاس‌ها، متغیرها و غیره) می‌باشند و می‌توان آن‌ها را در برنامه‌های دیگر وارد کرد<sup>۹۷</sup> و دوباره استفاده کرد. ماژول‌ها به سازماندهی منطقی کد، افزایش قابلیت استفاده مجدد و ساده‌تر شدن نگهداری کمک می‌کنند. این ماژول‌ها می‌توانند داخلی (از پیش فراهم‌شده توسط پایتون) یا تعریف‌شده توسط کاربر باشند.

### ۱-۱۲) چرا باید از ماژول‌ها استفاده کنید؟

۱. سازماندهی کد: برنامه‌های پیچیده را به فایل‌های کوچک‌تر و قابل مدیریت تقسیم کنید.
۲. قابلیت استفاده مجدد: یک‌بار کد بنویسید و در پروژه‌های مختلف از آن استفاده کنید.
۳. مدیریت فضای نام: از تداخل کدها جلوگیری کنید و هر کد را در یک ماژول جداگانه قرار دهید.

## ۲-۱۲) انواع ماژول‌ها

### ۱-۲-۱۲) ماژول‌های داخلی

این ماژول‌ها از پیش در پایتون نصب شده‌اند. و نیازی به نصب آن‌ها از طریق pip نیست. از جمله‌ی این ماژول‌ها می‌توان به random، os، math و sys اشاره کرد.

مثال:

---

*import math*

<sup>۹۷</sup> import

```
print(math.sqrt(16))
```

---

## ۲-۲-۱۲) مازول‌های تعریف شده توسط کاربر

ماژول‌هایی که توسط کاربران برای استفاده مجدد از کد سفارشی ایجاد شده است. به شکل ۱-۱۲ و ۲-۱۲ توجه نمایید.

python example - my\_module.py

```
1 # Modules in Python
2 # my_module.py
3
4 def greet(name):
5     return f"Hello, {name}!"
```

شکل ۱-۱۲-۱- کدهای مربوط به مازول *my\_module.py*

python example - main.py

```
1 # Modules in Python
2 # main.py
3
4 import my_module # وارد کردن مازول به داخل پروژه
5
6 print(my_module.greet("Reza")) # استفاده از مازول وارد شده به پروژه
```

شکل ۱-۱۲-۲- وارد کردن مازول *my\_module.py* در درون پروژه *main.py*

خروجی شکل ۱-۱۲:

---

**Hello, Reza!**

---

### ۳-۲-۱۲) مازول‌های شخص ثالث

مازول‌هایی که توسط خود شرکت سازنده‌ی پایتون ارائه شده است و از طریق `pip` نصب می‌شوند مانند `numpy` ... و `pandas`

---

*`pip install numpy`*  
*`pip install pandas`*  
*`pip install python_package`*

---

### ۳-۱۲) وارد کردن مازول‌ها به داخل پروژه

برای وارد کردن مازولی به داخل کدهایتان، روش‌های مختلفی وجود دارد که در ادامه به آن‌ها پرداخته می‌شود.

### ۱-۳-۱۲) وارد کردن پایه‌ای

در این روش ابتدا کلیدواژه `import` را نوشه و سپس نام مازول را بنویسید.

---

*`import module_name`*

---

### ۱۲-۳-۲) وارد کردن مقدار خاص مانند کلاس از یک مازول

در این روش، در مرحله اول کلیدواژه `from` را نوشت، در مرحله‌ی دوم نام مازول، در مرحله سوم کلیدواژه `import` و در مرحله چهارم نام کلاس‌ها یا مقادیری که در مازول وجود دارد، نوشت، می‌شود.

---

*`from module_name import member1, member2`*

---

### ۱۲-۳-۳) وارد کردن تمامی مقادیری که در داخل مازول وجود دارد

مراحل این روش همانند روش قبل می‌باشد با این تفاوت که مرحله‌ی چهارم این روش از ستاره (\*) استفاده می‌شود.

---

*`from module_name import *`*

---

### ۱۲-۳-۴) وارد کردن مازول با دادن نام مستعار برای آن مازول

در این روش، در مرحله اول کلیدواژه `import` را نوشت، در مرحله دوم نام مازول، در مرحله سوم، از کلیدواژه `as` استفاده کرده و در نهایت، در مرحله چهارم نام مستعار دلخواه را بنویسید.

---

*`import module_name as name_delkhah`*

---

نکته: در این روش، معمولاً برای دسترسی به کلاس‌ها و توابع وابسته مازول، به جای این‌که خود تابع را فراخوانی کنید، از نام مستعاری که تعریف کردید، به عنوان نام مازول باید استفاده کنید.

#### ۴-۱۲) ایجاد و استفاده از مازول

۱) ساخت فایل پایتون به عنوان مازول: فایل پایتون را با نام دلخواه (در اینجا از نام my\_module.py) ایجاد کرده و درون آن کدهای مربوط به مازول را قرار دهید. در اینجا کدهای شکل ۳-۱۲ به عنوان یک مازول قرار داده شده است.

python example - my\_module.py

```
9  # Modules in Python
10 # my_module.py
11
12 def add(a, b):
13     return a + b
14
15 def subtract(a, b):
16     return a - b
17
```

شکل ۳-۱۲- ایجاد فایل پایتونی my\_module.py به عنوان یک مازول

۲) یک فایل دیگر پایتون در همان پوشه در کنار فایل پایتون که در مرحله قبل ایجاد شد، ایجاد کرده و برای این‌که بتوان از فایل پایتون مرحله قبل به عنوان مازول در این فایل پایتون استفاده کنید نیازه که مازول را وارد کنید. سپس برای استفاده از توابع فایل my\_module.py طبق روال شکل عمل کنید.

### python example - main.py

```
10 # Modules in Python  
11 # main.py  
12  
13 import my_module # وارد کردن مژول  
14  
15 print(my_module.add(5, 3))  
16 print(my_module.subtract(5, 3))
```

شکل ۱۲-۴- استفاده از فایل پایتون *my\_module.py* به عنوان مژول در فایل پایتونی *main.py*

خروجی شکل ۱۲-۴:

---

8

2

---

### ۱۲-۵) متغیر name در پایتون

متغیر name یک متغیر داخلی و خاص در پایتون است که نشان‌دهنده‌ی نام یک مژول است. این متغیر به صورت خودکار توسط مفسر پایتون مقداردهی می‌شود و در تعیین اینکه یک مژول مستقیماً اجرا شده یا به عنوان مژول وارد شده است، کاربرد دارد.

### ۱-۵-۱۲) چگونگی عملکرد `__name__`

۱. اگر یک فایل پایتون مستقیماً اجرا شود، مقدار متغیر `__name__` در آن فایل برابر با "خواهد بود.
۲. اگر فایل پایتون به عنوان یک ماژول در برنامه دیگری وارد شود، مقدار `__name__` برابر با نام آن ماژول خواهد بود.

### ۲-۵-۱۲) کاربرد `__name__`

از این متغیر برای اجرای کدهایی که فقط باید در صورتی که فایل مستقیماً اجرا شود، استفاده می‌شود. این رویکرد برای جلوگیری از اجرای ناخواسته‌ی کدها هنگام وارد کردن ماژول‌ها ضروری است.

### ۳-۵-۱۲) مثال‌های مربوط به `__name__`

مثال ۱. فرض کنید یک فایل پایتون به نام `test.py` دارید:

```
python example - test.py
```

```
1 # Modules in Python
2
3 # test.py
4 def greet():
5     print("Hello, world!")
6
7 if __name__ == "__main__":
8     print("This file is executed directly.")
9     greet()
```

شکل ۱۲-۵-۱۲) مثال مربوط به استفاده از `__name__` در فایل اصلی

حالتهای اجرا:

۱. اجرای مستقیم فایل: اگر فایل را مستقیماً اجرا کنید:

---

*This file is executed directly.*

*Hello, world!*

---

۲. وارد کردن فایل به عنوان مازول: اگر فایل را در یک فایل دیگر مانند main.py وارد کنید همانند شکل :

python example - main.py

```
20 # Modules in Python  
21 # main.py  
22  
23 import test
```

شکل ۱۲-۶-وارد کردن مازول *test* به درون *main.py*

خروجی شکل ۱۲-۶:

هیچ کدام از کدهای داخل if \_\_name\_\_ == "\_\_main\_\_": اجرا نمی‌شوند.

مثال ۲. فرض کنید دو فایل دارید:

✓ فایل my\_module.py

python example - my\_module.py

```
20 # Modules in Python  
21 # my_module.py  
22  
23 def add(a, b):  
24     return a + b  
25  
26 if __name__ == "__main__":  
27     print("module1 is executed directly.")  
28     print(add(5, 3))
```

شكل ۱۲-۷- مثال مربوط به کدهای فایل *my\_module.py*

فایل main.py ✓

python example - main.py

```
27 # Modules in Python  
28 # main.py  
29  
30 import my_module  
31 print("main.py is running.")  
32 result = my_module.add(10, 20)  
33 print("Result:", result)
```

شكل ۱۲-۸- مثال مربوط به کدهای فایل *main.py*

نتایج اجرا:

❖ اجرای my\_module.py شکل ۱۲-۷-

---

*module1 is executed directly.*

---

❖ اجرای main.py یا شکل ۱۲-۸:

---

*main.py is running.*

---

**Result: 30**

---

در این مثال، کد داخل if \_\_name\_\_ == "\_\_main\_\_": در فایل my\_module.py هنگام اجرا نمی‌شود.

#### ۱۲-۵-۴) مزایای استفاده از \_\_name\_\_

۱. جلوگیری از اجرای ناخواسته: کدی که داخل if \_\_name\_\_ == "\_\_main\_\_": قرار می‌گیرد، فقط

در صورت اجرای مستقیم فایل اجرا می‌شود.

۲. قابلیت استفاده مجدد: یک فایل می‌تواند هم به عنوان یک ماژول (برای وارد کردن در برنامه‌های دیگر) و

هم به عنوان یک اسکریپت مستقل (برای اجرا) عمل کند.

۳. مدیریت بهتر تست‌ها: از این روش می‌توان برای اجرای تست‌ها در یک فایل استفاده کرد.

# فصل سیزدهم

آشنایی با مازول Random در پایتون

## ۱۳) آشنایی با مژول **Random** در پایتون

ماژول random در پایتون بخشی از کتابخانه استاندارد است که برای تولید اعداد تصادفی و انجام عملیات‌های مبتنی بر تصادف طراحی شده است. این ماژول بر اساس الگوریتم‌های تولید اعداد شبه‌تصادفی عمل می‌کند و ابزارهای مختلفی برای کاربردهای گوناگون، از شبیه‌سازی گرفته تا انتخاب‌های تصادفی در بازی‌ها یا مسائل آماری، فراهم می‌کند.

### ۱-۱۳) ویژگی‌های ماژول **random**

۱. تولید اعداد شبه‌تصادفی:

- اعداد تولیدشده توسط این ماژول به صورت شبه‌تصادفی هستند، به این معنا که با الگوریتم‌های خاصی تولید می‌شوند و در ظاهر تصادفی به نظر می‌رسند، اما در واقع از یک سری قواعد ریاضی پیروی می‌کنند.
- این الگوریتم‌ها معمولاً از دنباله‌ای ثابت (بر اساس دانه اولیه یا seed) شروع به کار می‌کنند، بنابراین می‌توان نتایج را در صورت نیاز تکرارپذیر کرد.

۲. منبع اصلی تولید اعداد:

- الگوریتم تولید اعداد تصادفی در random مبتنی بر مرسن پیچشی Mersenne Twister است، که یکی از پرکاربردترین و کارآمدترین الگوریتم‌ها برای تولید اعداد شبه‌تصادفی است.
- این الگوریتم دقیق‌ترین دقت بالایی در شبیه‌سازی دارد و اعداد تولیدی آن به خوبی از نظر آماری "تصادفی" رفتار می‌کنند.

۳. کاربردهای گسترده:

- این ماثول علاوه بر تولید اعداد تصادفی، ابزارهایی برای شبیه‌سازی توزیع‌های آماری مختلف (مانند توزیع نرمال، یکنواخت، نمایی و غیره) فراهم کرده است.

- امکان انتخاب یا مرتب‌سازی تصادفی داده‌ها در دنباله‌ها (مانند فهرست‌ها و رشته‌ها) را ارائه می‌دهد.

#### ۴. تکرارپذیری و کنترل تصادف:

- با استفاده از تابع `random.seed()` می‌توان رفتار تولید اعداد تصادفی را کنترل کرد. این قابلیت به‌ویژه در آزمایشات و شبیه‌سازی‌هایی که نیاز به نتایج تکرارپذیر دارند، بسیار مفید است.

### ۲-۱۳) محدودیت‌های ماثول `random`

۱. شبه‌تصادفی بودن: این ماثول برای تولید اعداد کاملاً تصادفی مناسب نیست، زیرا رفتار آن وابسته به الگوریتم‌های ریاضی است. برای تولید اعداد تصادفی واقعی (مانند اعداد مبتنی بر نویز سخت‌افزاری)، باید از کتابخانه‌هایی مانند `os.urandom()` یا `secrets` استفاده کرد.

۲. امنیت پایین: ماثول `random` برای اهداف امنیتی (مانند تولید رمزهای عبور) طراحی نشده است. در این موارد، استفاده از پکیج `secrets` توصیه می‌شود.

۳. سرعت محدود: برای شبیه‌سازی‌های بسیار بزرگ یا کاربردهایی که نیاز به سرعت فوق العاده دارند، ممکن است استفاده از کتابخانه‌های دیگر (مانند نامپای) ترجیح داده شود.

نکات:

- بخشی از کتابخانه استاندارد پایتون است و نیازی به نصب جداگانه ندارد.
- این ماثول اغلب به عنوان ابزار پیش‌فرض برای عملیات‌های ساده تصادفی استفاده می‌شود.

- برای نیازهای پیچیده‌تر، ماثول‌های جایگزین مانند نامپای (برای توزیع‌های پیچیده و عملیات ریاضی پیشرفت‌های امنیتی) در دسترس هستند.

### ۳-۱۳) توابع وابسته در ماثول **random**

در جدول ۱-۱۳ تمامی توابع وابسته موجود در ماثول **random** و توضیح کوتاه برای هر تابع وابسته آمده است:

جدول ۱۳-۱- جدول مربوط به تمامی توابع وابسته موجود در ماثول **random**

| توضیحات                                                       | تابع وابسته                            |
|---------------------------------------------------------------|----------------------------------------|
| تولید عدد اعشاری تصادفی در بازه $[0,1]$ .                     | random()                               |
| تولید عدد اعشاری تصادفی در بازه $[a, b]$ .                    | uniform(a, b)                          |
| تولید عدد صحیح تصادفی در بازه $[a, b]$ (شامل هر دو).          | randint(a, b)                          |
| تولید عدد صحیح تصادفی از دنباله‌ای با شروع، پایان و گام مشخص. | randrange(start, stop[, step])         |
| انتخاب یک عنصر تصادفی از دنباله (فهرست، تاپل، یا رشته).       | choice(seq)                            |
| انتخاب $k$ عنصر تصادفی از دنباله با امکان وزن‌دهی.            | choices(population, weights=None, k=1) |
| انتخاب تصادفی $k$ عنصر بدون جای‌گذاری از دنباله.              | sample(population, k)                  |
| مرتب‌سازی تصادفی عناصر فهرست (تغییر در جای خود).              | shuffle(seq)                           |
| تنظیم دانه (Seed) برای تکرارپذیری اعداد تصادفی.               | seed(a=None, version=2)                |
| دریافت وضعیت داخلی مولد تصادفی.                               | getstate()                             |
| تنظیم وضعیت داخلی مولد تصادفی به وضعیت خاص.                   | setstate(state)                        |
| تولید عدد تصادفی با توزیع بتا.                                | betavariate(alpha, beta)               |
| تولید عدد تصادفی با توزیع نمایی.                              | expovariate(lambd)                     |

|                                                                        |                             |
|------------------------------------------------------------------------|-----------------------------|
| تولید عدد تصادفی با توزیع گاما.                                        | gammavariate(alpha, beta)   |
| تولید عدد تصادفی با توزیع نرمال (گاوی) با میانگین و انحراف معیار مشخص. | gauss(mu, sigma)            |
| تولید عدد تصادفی با توزیع لگاریتمی نرمال.                              | lognormvariate(mu, sigma)   |
| تولید عدد تصادفی با توزیع نرمال (شبه نرمال).                           | normalvariate(mu, sigma)    |
| تولید عدد تصادفی با توزیع ون میزس (برای داده های زاویه ای).            | vonmisesvariate(mu, kappa)  |
| تولید عدد تصادفی با توزیع پارتلو.                                      | paretovariate(alpha)        |
| تولید عدد تصادفی با توزیع مثلثی.                                       | triangular(low, high, mode) |
| تولید عدد تصادفی با توزیع ویبول.                                       | weibullvariate(alpha, beta) |

## ۴-۱۳) کاربردهای اصلی و توابع مهم ماژول **random**

۱-۴-۱۳) تولید اعداد تصادفی با استفاده از ماژول **random**

۱-۱-۴-۱۳) تولید عددی تصادفی بین بازه **(۰,۱)**

---

### *random.random()*

---

python example - random\_package.py

```

1 # Random module in Python
2
3 import random # ماژول رندوم در پایتون
4
5 # random.random()
6 x = random.random()
7 print(x)
8

```

شکل ۱۳-۱- مثال مربوط به تولید عدد تصادفی بین بازه صفر تا یک با تابع وابسته **random**

خروجی شکل ۱-۱۳ :

---

0.9442700829400728

---

۱۳-۴-۲) تولید عددی اعشاری تصادفی بین a و b

---

*random.uniform(a, b)*

---

python example - random\_package.py

```
11 # Random module in Python مازول رندوم در پایتون
12
13 import random # وارد کردن مازول
14
15 # # random.uniform(a, b)
16 x = random.uniform(5,10)
17 print(x)
```

شکل ۱۳-۲- مثال مربوط به تولید عدد تصادفی بین عدد ۵ تا ۱۰ با استفاده از تابع وابسته *uniform*

خروجی شکل ۲-۱۳ :

---

8.714437773526399

---

۱۳-۴-۲) تولید اعداد صحیح

۱۳-۴-۲-۱) تولید عدد صحیح تصادفی  $a$  تا  $b$

---

*random.randint(a, b)*

---

python example - random\_package.py

```
21 # Random module in Python ماثول رندوم در پایتون
22
23 import random # وارد کردن ماثول
24
25 # random.randint(a, b)
26 x = random.uniform(5,10)
27 print(x)
```

شکل ۱۳-۳-مثال مربوط به تولید عدد تصادفی از عدد ۵ تا ۱۰ با استفاده از تابع وابسته *randint*

خروجی شکل ۱۳-۳:

---

6.15293232726932

---

۱۳-۴-۲-۲) تولید عدد صحیح تصادفی از دنباله‌ای با فاصله مشخص

---

*random.randrange(start, stop[, step])*

---

python example - random\_package.py

```
31 # Random module in Python ماثول رندوم در پایتون
32
33 import random # وارد کردن ماثول
34
35 # random.randrange(start, stop[, step])
36 print(random.randrange(2, 20, 4))
```

شکل ۱۳-۴-مثالی از تولید عدد صحیح تصادفی از دنباله‌ای با فاصله مشخص با استفاده از تابع *randrange*

خروجی شکل ۱۳-۴:

---

18

---

۱۳-۴-۳) کار با فهرست‌ها

۱۳-۴-۳-۱) انتخاب تصادفی یک عنصر از دنباله (فهرست، تاپل یا رشته)

---

*random.choice(seq)*

---

### python example - random\_package.py

```
40 # Random module in Python ماثول رندوم در پایتون
41
42 import random # وارد کردن ماثول
43
44 # random.choice(seq)
45 fruits = ['apple', 'banana', 'cherry']
46 print(random.choice(fruits))
```

شکل ۱۳-۵- مثالی از انتخاب تصادفی یک عنصر از دنباله با استفاده از تابع وابسته *choice*

خروجی شکل ۱۳-۵:

---

*banana*

---

۱۳-۴-۳-۲) انتخاب تصادفی k عنصر از دنباله با امکان تعریف وزن.

---

*random.choices(seq, weights=None, k=1)*

---

python example - random\_package.py

```
50 # Random module in Python مازول رندوم در پایتون
51
52 import random # مازول وارد کردن
53
54 # random.choices(seq, weights=None, k=1)
55 fruits = ['apple', 'banana', 'cherry']
56 print(random.choices(fruits, weights=[1, 3, 1], k=5))
```

شکل ۱۳-۶- مثالی از انتخاب تصادفی ۵ عنصر از دنباله با استفاده از تابع وابسته *choices*

خروجی شکل ۱۳-۶:

---

*['banana', 'banana', 'cherry', 'cherry', 'banana']*

---

۱۳-۴-۳-۳- چیدمان تصادفی عناصر یک فهرست

---

*random.shuffle(seq)*

---

python example - random\_package.py

```
60 # Random module in Python ماثول رندوم در پایتون
61
62 import random # وارد کردن ماثول
63
64 # random.shuffle(seq)
65 fruits = ['apple', 'banana', 'cherry']
66 random.shuffle(fruits)
67 print(fruits)
68
```

شکل ۱۳-۷-مثالی از چیدمان تصادفی عناصر یک فهرست با استفاده از تابع *shuffle*

خروجی شکل ۱۳-۷:

---

*['cherry', 'apple', 'banana']*

---

۱۳-۴-۳-۴ نمونه‌گیری از یک فهرست، انتخاب *k* عنصر تصادفی بدون جای‌گذاری از دنباله.

---

*random.sample(seq, k)*

---

python example - random\_package.py

```
71 # Random module in Python مژول رندوم در پایتون
72
73 import random # وارد کردن مژول
74
75 # random.sample(seq, k)
76 fruits = ['apple', 'banana', 'cherry']
77 print(random.sample(fruits, 2))
```

شکل ۱۳-۸- مثال مربوط به نمونه‌گیری از یک فهرست با انتخاب ۲ عنصر با استفاده از تابع وابسته *sample*

خروجی شکل ۱۳-۸:

---

*['apple', 'cherry']*

---

۱۳-۴-۴) توابع توزیع آماری

۱۳-۴-۴-۱) تولید عدد تصادفی با توزیع نرمال (گاوی).

---

*random.gauss(mu, sigma)*

---

python example - random\_package.py

```
81 # Random module in Python  
82  
83 import random # ماژول  
84  
85 # random.gauss(mu, sigma)  
86 print(random.gauss(0, 1))
```

شکل ۹-۱۳- مثال مربوط به تولید عدد تصادفی با توزیع گاووسی

خروجی شکل ۹-۱۳:

---

-0.04855224499778251

---

۱۳-۴-۴-۲) تولید عدد تصادفی با توزیع نمایی.

---

*random.expovariate(lambd)*

---

python example - random\_package.py

```
90 # Random module in Python  
91  
92 import random # ماژول  
93  
94 # random.expovariate(lambd)  
95 print(random.expovariate(1.5))
```

شکل ۱۰-۱۳- مثال مربوط به تولید عدد تصادفی با توزیع نمایی

خروجی شکل ۱۰-۱۳:

---

0.7065978465693531

---

۱۳-۴-۴-۳) تولید عدد تصادفی با توزیع بتا.

---

*random.betavariate(alpha, beta)*

---

python example - random\_package.py

```
99 # Random module in Python
100
101 import random # ماژول رندوم در پایتون
102
103 # random.betavariate(alpha, beta)
104 print(random.betavariate(2, 5))
```

شکل ۱۱-۱۳-۱۱- مثال مربوط به تولید عدد تصادفی با توزیع بتا

خروجی شکل ۱۱-۱۳:

---

0.347542255508811

---

: نکته

تعیین دانه<sup>۹۸</sup>: برای تولید اعداد تصادفی تکرارپذیر (برای مثال در آزمایشات)، از random.seed استفاده کنید.

python example - random\_package.py

```
108 # Random module in Python  
109  
110 import random # ماژول رندوم در پایتون  
111  
112 # Seed in Random module  
113 random.seed(42)  
114 print(random.random()) # خروجی همواره ثابت خواهد بود
```

شکل ۱۲-۱۳- تولید اعداد تصادفی تکرارپذیر با تابع وابسته *seed*

خروجی شکل ۱۲-۱۳:

خروجی برای بار اول:

---

0.6394267984578837

---

خروجی برای باز دوم:

---

0.6394267984578837

---

<sup>98</sup> Seed

# فصل چهاردهم

آشنایی با ماثول Math در پایتون

## ۱۴) آشنایی با مژول **math** در پایتون

ماژول **math** در پایتون یکی از ماژول‌های داخلی و پایه‌ای است که برای انجام محاسبات ریاضی پیشرفته طراحی شده است. این ماژول شامل توابع و ثابت‌هایی است که نیازهای گسترده‌ای از عملیات‌های ریاضی مانند مثلثاتی، لگاریتمی، توابع خاص، و تبدیل‌های زاویه را پوشش می‌دهد. **math** به دلیل کارایی و دقت بالا، در بسیاری از برنامه‌های علمی، مهندسی و داده‌کاوی مورد استفاده قرار می‌گیرد.

### ۱-۱۴) ویژگی‌های ماژول **math**

۱. تابع عمومی ریاضی: این ماژول شامل تابع پایه‌ای مانند جمع، تفریق، ضرب و تقسیم نیست (زیرا این عملیات‌ها مستقیماً در پایتون پشتیبانی می‌شوند)، اما تابع پیچیده‌تری مانند محاسبه قدر مطلق، توان، جذر، و تبدیل اعداد ارائه می‌دهد. همچنین قابلیت‌های دیگری مانند گرد کردن اعداد به بالا یا پایین و محاسبات مرتبط با اعداد گویا و فاکتوریل را فراهم می‌کند.
۲. تابع مثلثاتی و زاویه‌ای: تابع مثلثاتی استاندارد مانند سینوس، کسینوس، تانژانت و معکوس آن‌ها در این ماژول وجود دارد. علاوه بر این، توابعی برای تبدیل بین درجه و رادیان و همچنین محاسبات مثلثاتی معکوس برای حل معادلات پیچیده‌تر فراهم شده است.
۳. تابع لگاریتمی و نمایی: مژول **math** ابزارهایی برای محاسبه لگاریتم در پایه‌های مختلف (مانند پایه ۲ یا پایه ۱۰) و همچنین تابع نمایی برای توان‌های طبیعی یا عدد نپر ارائه می‌دهد. این تابع برای محاسباتی که شامل رشد یا کاهش نمایی، شبیه‌سازی یا مدل‌سازی هستند، بسیار مفید است.
۴. ثابت‌های ریاضی: این ماژول شامل ثابت‌های ریاضی معروفی مانند عدد  $\pi$  و عدد نپر است که به دقت بالا در محاسبات علمی کمک می‌کند. همچنین، ثابت‌های دیگری مانند تاو، که دو برابر عدد  $\pi$  است.

۵. توابع آماری و اعداد تصادفی: گرچه `math` به طور مستقیم برای تولید اعداد تصادفی طراحی نشده است این کار توسط مازول `random` انجام می‌شود، توابع آماری مانند میانگین هندسی و ابزارهای مرتبط با اعداد اول و ترکیبات در این مازول قرار دارند

۶. مازول `math` به طور گسترده از توابع ریاضی استاندارد کتابخانه (libm) C استفاده می‌کند. این بدان معناست که بسیاری از توابع موجود در این مازول، مستقیماً با استفاده از کتابخانه ریاضی زبان C پیاده‌سازی شده‌اند. این طراحی دو مزیت کلیدی دارد:

➤ سرعت بالا: توابع این مازول، به دلیل استفاده از کتابخانه‌های سطح پایین و بهینه‌سازی شده، بسیار سریع هستند.

➤ قابلیت اطمینان: این توابع از استانداردهای IEEE 754 پیروی می‌کنند، که دقت محاسبات در اعداد ممیز شناور را تضمین می‌کند.

## ۲-۱۴) کاربردهای مازول `math`

مازول `math` برای برنامه‌هایی که نیاز به دقت بالا در محاسبات ریاضی دارند، ایده‌آل است. این مازول در بسیاری از حوزه‌ها مانند فیزیک، مهندسی، علوم داده، شبیه‌سازی و مدل‌سازی استفاده می‌شود. طراحی ساده و در عین حال قدرتمند آن، باعث شده تا به عنوان یک ابزار استاندارد برای توسعه‌دهندگان و پژوهشگران شناخته شود. به دلیل تنوع توابع و ثابت‌های ارائه شده، این مازول می‌تواند نیازهای محاسباتی در بسیاری از رشته‌ها و پژوهش‌ها را پوشش دهد. در ادامه برخی از کاربردهای کلیدی آن آورده شده است:

۱. علوم ریاضی و مهندسی
  - انجام محاسبات پیچیده ریاضی مانند جذر، لگاریتم، توان، و محاسبات مرتبط با توابع مثلثاتی.

• مدل سازی ریاضی و حل معادلات شامل توزیع های خاص، توابع لگاریتمی، و توابع مثلثاتی معکوس.

• طراحی سیستم های کنترل، شبیه سازی های عددی و تحلیل داده ها.

## ۲. گرافیک کامپیوتری و بازی سازی

• استفاده از توابع مثلثاتی برای محاسبات مرتبط با چرخش، مقیاس دهی و حرکت اشیاء در فضای سه بعدی.

• محاسبه فاصله بین نقاط، بردارها، یا زوایا در بازی ها یا شبیه سازی های گرافیکی.

• تنظیمات دوربین و نور پردازی در گرافیک سه بعدی با استفاده از توابع مثلثاتی.

## ۳. علوم داده و تحلیل آماری

• پردازش داده ها و استفاده از توابع لگاریتمی و نمایی برای تغییر مقیاس داده ها.

• محاسبه مقادیر میانگین هندسی و استفاده از توابع توزیع خاص برای شبیه سازی داده ها.

• استفاده از ثابت های ریاضی مانند  $\pi$  برای تحلیل های هندسی.

## ۴. شبیه سازی های علمی

• محاسبات مرتبط با موج ها، ارتعاشات یا نوسانات با استفاده از توابع مثلثاتی و نمایی.

• شبیه سازی رفتار سیستم های دینامیکی با تابع مرتبط با لگاریتم یا توزیع های خاص.

## ۵. هوش مصنوعی و یادگیری ماشین

• استفاده در پیش پردازش داده ها مانند نرم افزاری با تابع  $\log$  یا محاسبات توان.

• پیاده سازی الگوریتم های مبتنی بر هندسه یا بهینه سازی.

• محاسبات فاصله (مانند فاصله اقلیدسی) برای الگوریتم های دسته بندی یا خوش بندی.

## ۶. فیزیک و نجوم

• استفاده در محاسبات مرتبط با مکانیک کلاسیک، نسبیت، یا دینامیک سیالات.

• محاسبه زوایا، سرعت‌ها، و بردارها در مسائل مکانیکی.

• شبیه‌سازی حرکت سیارات یا پیش‌بینی مسیر اجرام آسمانی با استفاده از ثابت‌های ریاضی و توابع مثلثاتی.

## ۷. مهندسی نرم‌افزار و رمزنگاری

• محاسبات عددی دقیق برای الگوریتم‌های رمزنگاری.

• تولید اعداد شبه‌تصادفی با توزیع‌های خاص برای تست یا الگوریتم‌های امنیتی.

• تحلیل‌های عددی در پیاده‌سازی الگوریتم‌های پیچیده.

## ۸. مهندسی مکانیک و عمران

• طراحی سازه‌ها و تحلیل استاتیکی و دینامیکی با استفاده از توابع مثلثاتی و لگاریتمی.

• محاسبه طول‌ها، زوایا و حجم‌ها در طراحی مکانیکی یا مهندسی ساختمان.

• شبیه‌سازی رفتار مواد در شرایط خاص.

## ۹. زیست‌شناسی و علوم زیستی

• محاسبات مرتبط با رشد جمعیت یا مدل‌سازی فرآیندهای زیستی با توابع نمایی یا لگاریتمی.

• تحلیل داده‌های ژنتیکی یا شبیه‌سازی فرایندهای فرآیندهای تکاملی.

• استفاده در مدل‌سازی‌های زیست‌محیطی یا تغییرات اقلیمی.

## ۱۴-۳) توابع وابسته موجود در ماژول **math**

در جدول ۱-۱۴، تمامی توابع وابسته‌ی ماژول **math** همراه با توضیح آن‌ها آورده شده است:

**جدول ۱-۱۴ - جدول مربوط به تمامی توابع وابسته‌ی موجود در ماژول **Math****

| توضیحات                                                                  | تابع وابسته                |
|--------------------------------------------------------------------------|----------------------------|
| مقدار عدد $x$ را به نزدیک‌ترین عدد صحیح بزرگ‌تر گرد می‌کند.              | ceil( $x$ )                |
| مقدار عدد $x$ را به نزدیک‌ترین عدد صحیح کوچک‌تر گرد می‌کند.              | floor( $x$ )               |
| مقدار مطلق عدد $x$ را برمی‌گرداند (بدون در نظر گرفتن علامت).             | fabs( $x$ )                |
| فاکتوریل عدد صحیح $x$ را محاسبه می‌کند.                                  | factorial( $x$ )           |
| مجموع دقیق عناصر یک iterable با دقت بالا را برمی‌گرداند.                 | fsum(iterable)             |
| بزرگ‌ترین مقسوم‌علیه مشترک (ب.م.م) دو عدد صحیح $a$ و $b$ را برمی‌گرداند. | gcd( $a, b$ )              |
| جذر عدد صحیح $n$ را به صورت عدد صحیح برمی‌گرداند.                        | isqrt( $n$ )               |
| جذر عدد $x$ را محاسبه می‌کند.                                            | sqrt( $x$ )                |
| مقدار $e^x$ (عدد نپر به $x$ ) را برمی‌گرداند.                            | exp( $x$ )                 |
| مقدار $1 - e^x$ را با دقت بالا برای مقادیر کوچک $x$ محاسبه می‌کند.       | expm1( $x$ )               |
| لگاریتم عدد $x$ را در پایه مشخص شده (پیش‌فرض: پایه $e$ ) محاسبه می‌کند.  | log( $x[, base]$ )         |
| لگاریتم عدد $x$ در پایه $10$ را محاسبه می‌کند.                           | log10( $x$ )               |
| لگاریتم عدد $x$ در پایه $2$ را محاسبه می‌کند.                            | log2( $x$ )                |
| مقدار $y^{**}x$ را برمی‌گرداند.                                          | pow( $x, y$ )              |
| حاصل ضرب عناصر یک iterable را محاسبه می‌کند.                             | prod(iterable, *, start=1) |
| قسمت اعشاری و صحیح عدد $x$ را به صورت دوتایی بازمی‌گرداند.               | modf( $x$ )                |
| بخش صحیح عدد $x$ را با حذف قسمت اعشاری برمی‌گرداند.                      | trunc( $x$ )               |

|                                                                |                                     |
|----------------------------------------------------------------|-------------------------------------|
| مقدار سینوس زاویه $x$ (بر حسب رادیان) را محاسبه می کند.        | $\sin(x)$                           |
| مقدار کسینوس زاویه $x$ (بر حسب رادیان) را محاسبه می کند.       | $\cos(x)$                           |
| مقدار تانژانت زاویه $x$ (بر حسب رادیان) را محاسبه می کند.      | $\tan(x)$                           |
| سینوس معکوس عدد $x$ را بر حسب رادیان برمی گردداند.             | $\text{asin}(x)$                    |
| کسینوس معکوس عدد $x$ را بر حسب رادیان برمی گردداند.            | $\text{acos}(x)$                    |
| تانژانت معکوس عدد $x$ را بر حسب رادیان برمی گردداند.           | $\text{atan}(x)$                    |
| زاویه قطبی (آرکتانژانت) برای مختصات $(y, x)$ را برمی گردداند.  | $\text{atan2}(y, x)$                |
| مقدار زاویه $x$ را از رادیان به درجه تبدیل می کند.             | $\text{degrees}(x)$                 |
| مقدار زاویه $x$ را از درجه به رادیان تبدیل می کند.             | $\text{radians}(x)$                 |
| طول هیپوتونوس (وتر) برای مجموعه ای از مختصات را محاسبه می کند. | $\text{hypot}(*\text{coordinates})$ |
| تابع گاما برای عدد $x$ را محاسبه می کند.                       | $\text{gamma}(x)$                   |
| لگاریتم طبیعی تابع گاما برای عدد $x$ را محاسبه می کند.         | $\text{lgamma}(x)$                  |
| مقدار تابع خطای گاوسی برای عدد $x$ را برمی گردداند.            | $\text{erf}(x)$                     |
| مقدار تابع خطای مکمل گاوسی برای عدد $x$ را برمی گردداند.       | $\text{erfc}(x)$                    |
| مقدار عدد پی را ذخیره می کند.                                  | $\pi$                               |
| مقدار عدد نیپر ( $e$ ) را ذخیره می کند.                        | $e$                                 |
| مقدار عدد تاو را ذخیره می کند.                                 | $\tau$                              |
| مقدار بینهایت مشبت را ذخیره می کند.                            | $\infty$                            |
| مقدار Not a Number (NaN) را ذخیره می کند.                      | $\text{nan}$                        |

## ۱۴-۴) توابع مهم و پر کاربرد ماثول

۱-۱-۴-۱۴) تابع وابسته‌ی محاسبه رادیکال یک عدد

---

*math.sqrt(x)*

---

python example - math\_module.py

```
1 # math module in Python
2
3 import math # ماثول ریاضی در پایتون
4
5 # math.sqrt(x)
6 print(f"Square root of 225 is: {math.sqrt(225)}")
7 print(f"Square root of 1024 is: {math.sqrt(1024)}")
8 print(f"Square root of 1 is: {math.sqrt(1)}")
9 print(f"Square root of 2 is: {math.sqrt(2)}")
```

شکل ۱-۱۴-۱- مثالی از نحوه محاسبه رادیکال با ماثول *math*

خروجی شکل ۱-۱۴:

---

*Square root of 225 is: 15.0*

*Square root of 1024 is: 32.0*

*Square root of 1 is: 1.0*

*Square root of 2 is: 1.414213562373095*

---

۱۴-۴-۲) تابع وابسته‌ی محاسبه توان یک عدد.

---

### *math.pow(x, y)*

---

python example - math\_module.py

```
13 # math module in Python
14
15 import math # وارد کردن مازول
16
17 # math.pow(x, y)
18 print(f"2 ** 3 = {math.pow(2, 3)}")
19 print(f"5 ** 2 = {math.pow(5, 2)}")
20 print(f"1 ** 100000 = {math.pow(1, 100000)}")
21 print(f"0 ** 5 = {math.pow(0, 5)}")
22 print(f"5 ** 5 = {math.pow(5, 5)}")
```

شکل ۱۴-۲-مثالی از محاسبه توان یک عدد با مازول *math*

خروجی شکل ۱۴-۲:

*2 \*\* 3 = 8.0*

*5 \*\* 2 = 25.0*

*1 \*\* 100000 = 1.0*

*0 \*\* 5 = 0.0*

*5 \*\* 5 = 3125.0*

---

۱۴-۳) تابع وابسته‌ی محاسبه فاکتوریل یک عدد صحیح.

---

*math.factorial(x)*

---

python example - math\_module.py

```
26 # math module in Python
27
28 import math # وارد کردن ماثول
29
30 # math.factorial(x)
31 print(f"Factorial number 5: {math.factorial(5)}")
32 print(f"Factorial number 10: {math.factorial(10)}")
33 print(f"Factorial number 3: {math.factorial(3)}")
34 print(f"Factorial number 1: {math.factorial(1)}")
```

شکل ۱۴-۳- مثالی از محاسبه فاکتوریل یک عدد با ماثول *math*

خروجی شکل ۱۴-۳:

---

*Factorial number 5: 120*

*Factorial number 10: 3628800*

*Factorial number 3: 6*

*Factorial number 1: 1*

---

۴-۱-۴) تابع وابسته‌ی محاسبه مجموع دقیق عناصر یک فهرست.

---

### ***math.fsum(iterable)***

---

python example - math\_module.py

```
38 # math module in Python مازول ریاضی در پایتون
39
40 import math # وارد کردن مازول
41
42 numbers_list_1 = [29, 54, 7, 47, 98, 2, 36, 813, 9, 35, 52, 461, 1, 41, 92]
43 numbers_list_2 = [68, 43, 21, 458, 42, 7, 75, 1, 92, 95, 8, 56, 242, 58, 8]
44
45 # math.fsum(iterable)
46 print(f"Sum of numbers_list_1 is: {math.fsum(numbers_list_1)}")
47 print(f"Sum of numbers_list_2 is: {math.fsum(numbers_list_2)}")
```

شکل ۴-۱۴-۴) مثالی از محاسبه مجموع اعداد یک فهرست با استفاده از تابع وابسته *fsum*

خروجی شکل ۴-۱۴:

---

*Sum of numbers\_list\_1 is: 1777.0*

*Sum of numbers\_list\_2 is: 1274.0*

---

۴-۱-۲-۴) توابع مثلثاتی

۱-۲-۴-۱) توابع وابسته‌ی محاسبه سینوس، کسینوس و تانژانت

---

### ***math.sin(x)***

*math.cos(x)*

*math.tan(x)*

---

python example - math\_module.py

```
51 # math module in Python مازول ریاضی در پایتون
52
53 import math # وارد کردن مازول
54
55 # math.sin(radian)
56 x = math.pi / 2 # (pi = 180 degree)
57 print(math.sin(x))
58
59 # math.cos(radian)
60 y = math.pi # (pi = 180 degree)
61 print(math.cos(y))
62
63 # math.tan(radian)
64 z = (math.pi / 4) # (pi = 180 degree)
65 print(math.tan(z))
```

شکل ۱۴-۵- مثال‌های مربوط به سینوس، کسینوس و تانژانت با استفاده از مازول *math*

:۵-۱۴ خروجی شکل

---

**1.0**

**-1.0**

**0.9999999999999999**

---

## ۱۴-۴-۲-۲) توابع وابسته‌ی محاسبه سینوس معکوس، کسینوس معکوس و تانژانت معکوس

*math.asin(x)*

*math.acos(x)*

*math.atan(x)*

python example - math\_module.py

```
ماژول ریاضی در پایتون  
82 # math module in Python  
83  
84 import math # ماژول  
85  
86 # math.asin(radian)  
87 print(math.asin(1))  
88  
89 # math.acos(radian)  
90 print(math.acos(1))  
91  
92 # math.atan(radian)  
93 print(math.atan(1))
```

شکل ۱۴-۶- مثال‌های مربوط به محاسبه سینوس معکوس، کسینوس معکوس و تانژانت معکوس توسط ماژول *math*

خروجی شکل ۱۴-۶:

1.5707963267948966

0.0

0.7853981633974483

---

*math.degrees(x)*

---

python example - math\_module.py

```
97 # math module in Python
98
99 import math # مازول ریاضی در پایتون
100
101 # Radian to Degree تبدیل رادیان به درجه
102 # math.degrees(x)
103
104 radian_list = [math.pi, math.pi/2, math.pi/4]
105
106 for i in radian_list:
107     print(f"Degree of {i} is {math.degrees(i)}")
```

شکل ۱۴-۷- مثال مربوط به تبدیل رادیان به درجه با استفاده از مازول *math*

خروجی شکل ۷-۱۴

---

*Degree of 3.141592653589793 is 180.0*

*Degree of 1.5707963267948966 is 90.0*

*Degree of 0.7853981633974483 is 45.0*

---

### *math.radians(x)*

---

python example - math\_module.py

```
111 # math module in Python  
112  
113 import math # مازول ریاضی در پایتون  
114  
115 # Degree to Radian تبدیل درجه به رادیان  
116 # math.radians(x)  
117  
118 degree_list = [180, 90, 45]  
119  
120 for i in degree_list:  
121     print(f"Radian of {i} is {math.radians(i)}")
```

شکل ۱۴-۸-مثال مربوط به تبدیل درجه به رادیان با استفاده از مازول *math*

خروجی شکل ۸-۱۴:

---

*Radian of 180 is 3.141592653589793*

*Radian of 90 is 1.5707963267948966*

*Radian of 45 is 0.7853981633974483*

---

#### ۳-۴-۱۴) توابع نمایی و لگاریتم‌ها

##### ۱-۳-۴-۱۴) تابع وابستهٔ محاسبه توابع نمایی

*math.exp(x)*

python example - math\_module.py

```
125 # math module in Python مازول ریاضی در پایتون
126
127 import math # وارد کردن مازول
128
129 # math.exp(x) محاسبه توابع نمایی
130 print(math.exp(65))
131 print(math.exp(-6.89))
```

شکل ۱۴-۹- مثال مربوط به محاسبه توابع نمایی با استفاده از مازول *math*

خروجی شکل ۹-۱۴:

*1.6948892444103338e+28*

*0.0010179138409954387*

#### ۲-۳-۴-۱۴) تابع وابستهٔ محاسبه لگاریتم یک عدد

*math.log(x[, base])*

python example - math\_module.py

```
ماژول ریاضی در پایتون # math module in Python
135
136
137 import math # وارد کردن ماژول
138
139 # math.log(x[, base]) محاسبه لگاریتم با اعمال مبنا
140 print (math.log(14))
141 print (math.log(14,5))
```

شکل ۱۰-۱۴-۱۰-مثال مربوط به محاسبه لگاریتم یک عدد با اعمال مبنای دلخواه با استفاده از ماژول *math*

خروجی شکل ۱۰-۱۴:

---

2.6390573296152584

---

1.6397385131955606

---

۱۰-۱۴-۳-۴-۳-تابع وابسته‌ی محاسبه لگاریتم یک عدد بر پایه‌ی

---

*math.log10(x)*

---

### python example - math\_module.py

```
145 # math module in Python ماثول ریاضی در پایتون
146
147 import math # وارد کردن ماثول
148
149 # math.log10(x) محاسبه لگاریتم با مبنای ۱۰
150 print(math.log10(10))
151 print(math.log10(100))
152 print(math.log10(1000))
153 print(math.log10(10000))
```

شکل ۱۴-۱۱-۱۴-مثال مربوط به محاسبه لگاریتم یک عدد با مبنای ۱۰ با استفاده از ماثول *math*

خروجی شکل ۱۱-۱۴:

---

1.0

2.0

3.0

4.0

---

۱۴-۴-۳-۲-۱۴) تابع وابستهٔ محاسبه لگاریتم یک عدد بر پایهٔ ۲.

---

*Math.log2(x)*

---

## python example - math\_module.py

```
157 # ریاضی ماذول در پایتون
158
159 import math # ماذول کردن
160
161 # لگاریتم 2 مبنای x محاسبه
162 print(math.log2(2))
163 print(math.log2(8))
164 print(math.log2(1024))
165 print(math.log2(64))
```

شکل ۱۴-۱۲- مثال مربوط به محاسبه لگاریتم یک عدد با مبنای ۲ با استفاده از مازول  $math$

خروجی شکل ۱۴-۱۲:

1.0

3.0

10.0

6.0

#### ١٤-٤) مدیریت اعداد اعشاری و صحیح

۱-۴-۴-۱) تابع وابسته‌ی گرد کردن عددی به سمت عدد بزرگ‌تر.

*math.ceil(x)*

### python example - math\_module.py

```
169 # math module in Python  
170  
171 import math # مازول  
172  
173 # ceil(x) به سمت عدد بزرگتر از خودش گردکردن عددی  
174 print(math.ceil(3.8))  
175 print(math.ceil(4.2))  
176 print(math.ceil(1.1))  
177 print(math.ceil(10.26))
```

شکل ۱۴-۱۳- مثال مربوط به گردکردن عددی به سمت عدد بزرگتر از خودش توسط مازول *math*

خروجی شکل ۱۴-۱۳:

---

4

5

2

11

---

۱۴-۱۳-۴-۲) تابع وابسته‌ی گرد کردن عددی به سمت عدد کوچک‌تر.

---

*math.floor(x)*

---

python example - math\_module.py

```
ماژول ریاضی در پایتون # math module in Python
181
182
183 import math # وارد کردن ماژول
184
185 # floor() گردکردن عددی به سمت عدد کوچکتر از خودش
186 print(math.floor(3.8))
187 print(math.floor(4.2))
188 print(math.floor(1.1))
189 print(math.floor(10.26))
```

**شکل ۱۴-۱۴**- مثال مریوط به گرد کردن عددی به سمت عدد کوچک تر از خودش توسط مازول *math*

خروجی شکل ۱۴-۱۴:

3

4

1

10

۱۴-۴-۳) تابع وابسته حذف قسمت اعشاری یک عدد اعشاری.

*math.trunc(x)*

python example - math\_module.py

```
ماژول ریاضی در پایتون # math module in Python
193
194
195 import math # وارد کردن ماژول
196
197 حذف قسمت اعشاری عدد # trunc(x)
198 print(math.trunc(1.25))
199 print(math.trunc(2.55216111862331))
200 print(math.trunc(10.00))
201 print(math.trunc(5.666666666666))
202 print(math.trunc(0))
```

شکل ۱۴-۱۵- مثال مربوط به حذف قسمت اعشاری یک عدد با استفاده از ماژول *math*

خروجی شکل ۱۴-۱۵ :

---

1

2

10

5

0

---

۱۴-۱۴-۱۴-۱۴) تابع وابسته جداسازی قسمت صحیح و اعشاری یک عدد اعشاری.

---

*math.modf(x)*

---

### python example - math\_module.py

```
206 # math module in Python مازول ریاضی در پایتون
207
208 import math # وارد کردن مازول
209
210 # modf(x) قسمت اعشاری و صحیح یک عدد
211 print(math.modf(10.00))
212 print(math.modf(1.25))
213 print(math.modf(1000000.5))
214 print(math.modf(0.258))
215 print(math.modf(5))
216 print(math.modf(21.123))
```

شکل ۱۶-۱۴- مثالی از جداسازی قسمت اعشاری و صحیح یک عدد توسط مازول *math*

خروجی شکل ۱۶-۱۴:

---

(0.0, 10.0)

(0.25, 1.0)

(0.5, 1000000.0)

(0.258, 0.0)

(0.0, 5.0)

(0.123, 21.0)

---

---

*math.pi*

---

python example - math\_module.py

```
220 # math module in Python  
221  
222 import math # مازول ریاضی در پایتون  
223  
224 # math.pi عدد پی  
225 print(math.pi)
```

شکل ۱۴-۱۷- عدد پی در مازول *math*

خروجی شکل ۱۷-۱۴:

---

3.141592653589793

---

---

*math.e*

---

python example - math\_module.py

```
229 # math module in Python ماثول ریاضی در پایتون
230
231 import math # وارد کردن ماثول
232
233 # math.e عدد نپر
234 print(math.e)
```

شکل ۱۴-۱۸ - عدد نپر در ماثول *math*

: ۱۸-۱۴ خروجی

---

2.718281828459045

---

۱۴-۱۳-۵-۴ عدد تاو (دو برابر مقدار عدد پی)

---

*math.tau*

---

python example - math\_module.py

```
238 # math module in Python ماثول ریاضی در پایتون
239
240 import math # وارد کردن ماثول
241
242 # math.tau عدد تاو
243 print(math.tau)
```

شکل ۱۹-۱۴ - عدد تاو در ماثول *math*

خروجی شکل ۱۴-۱۹ :

---

6.283185307179586

---

۱۴-۱۵-۴-۵) مقدار بی‌نهایت

---

*math.inf*

---

python example - math\_module.py

```
ماژول ریاضی در پایتون # math module in Python
247
248
249 وارد کردن ماژول import math #  
250
251 مقدار بی‌نهایت # math.inf
252 print(math.inf)
```

شکل ۱۴-۲۰- مقدار بی‌نهایت در ماژول *math*

خروجی شکل ۱۴-۲۰ :

---

*inf*

---

## ۵-۱۴) مثال‌هایی از کاربردهای عملی

۱. محاسبه زاویه‌ها در فیزیک یا مهندسی: استفاده از توابع مثلثاتی برای محاسبه زوایا یا نیروهای مکانیکی.
۲. تبدیل داده‌ها در تحلیل داده‌ها: استفاده از الگاریتم‌ها برای نرمال‌سازی یا مقیاس‌دهی داده‌ها.
۳. محاسبات آماری و اقتصادی: محاسبه مجموع یا حاصل‌ضرب مقادیر در تحلیل‌های آماری.
۴. شبیه‌سازی و مدل‌سازی علمی: استفاده از توابع نمایی و خاص برای شبیه‌سازی رفتارهای طبیعی.

# فصل پانزدهم

آشنایی با مازول Datetime در پایتون

## ۱۵ آشنایی با ماثول Datetime

ماژول `datetime` در پایتون یکی از ابزارهای اصلی برای کار با تاریخ و زمان است که شامل کلاس‌های متعددی برای مدیریت و پردازش داده‌های مرتبط با زمان و تاریخ می‌شود. یکی از این کلاس‌ها، کلاس `date` است که به طور خاص برای کار با تاریخ (بدون در نظر گرفتن زمان) طراحی شده است.

کلاس `date` امکان ایجاد و مدیریت تاریخ‌های خاص را فراهم می‌کند. این کلاس از سه مقدار اصلی، یعنی سال، ماه و روز استفاده می‌کند. این مقادیر بر اساس استانداردهای تقویمی میلادی تعریف می‌شوند و می‌توانند تاریخ‌های گذشته، حال یا آینده را مدیریت کنند. این قابلیت برای برنامه‌هایی که نیاز به پردازش تاریخ‌های دقیق دارند، مانند سیستم‌های تقویمی، گزارش‌دهی یا تحلیل داده‌ها، بسیار مفید است.

این کلاس از ویژگی‌های داخلی و توابع کاربردی برخوردار است که به کاربران اجازه می‌دهد تاریخ‌ها را استخراج، مقایسه و قالب‌بندی کنند. به عنوان مثال، می‌توان اطلاعات مربوط به سال، ماه یا روز را جداگانه دریافت کرد یا مقادیر تاریخ را برای بررسی‌های منطقی مقایسه کرد. علاوه بر این، کلاس `date` با ماثول‌های دیگر مانند `time` و `timedelta` تعامل دارد که برای انجام محاسبات مرتبط با زمان‌بندی یا فواصل زمانی مفید است.

از دیگر قابلیت‌های کلاس `date` می‌توان به توانایی تولید تاریخ فعلی، محاسبه فواصل زمانی میان دو تاریخ و ارائه تاریخ در قالب‌های قابل‌خواندن اشاره کرد. این امکانات باعث می‌شود که این کلاس در طیف گسترده‌ای از برنامه‌های کاربردی، از مدیریت زمان‌بندی پروژه‌ها گرفته تا تحلیل داده‌های تاریخی، مورد استفاده قرار گیرد.

ماژول `datetime` و کلاس `date` در کنار یکدیگر ابزارهای قدرتمندی برای مدیریت تاریخ و زمان در پایتون ارائه می‌دهند. این ابزارها علاوه بر سادگی، دقت و انعطاف‌پذیری بالایی دارند و با استانداردهای بین‌المللی سازگار هستند. این ویژگی‌ها موجب می‌شوند که ماژول `datetime` یکی از ماژول‌های پرکاربرد در پایتون باشد.

ماژول `datetime` در پایتون برای مدیریت و پردازش اطلاعات مرتبط با تاریخ و زمان بسیار قدرتمند است. این ماژول در طیف وسیعی از کاربردها مورد استفاده قرار می‌گیرد. در ادامه برخی از کاربردهای اصلی آن آورده شده است.

ماژول `datetime` در پایتون ویژگی‌های متنوع و قدرتمندی دارد که به شما کمک می‌کند تا با تاریخ و زمان به شیوه‌ای دقیق و انعطاف‌پذیر کار کنید. در ادامه، برخی از ویژگی‌های کلیدی این ماژول آورده شده است:

### ۱-۱۵) ویژگی‌های ماژول `datetime`

۱. کلاس‌های مختلف: ماژول `datetime` شامل چندین کلاس اصلی مانند `time`, `date`, `datetime`

۲. ایجاد و مدیریت تاریخ‌ها و زمان‌ها

- کلاس `date` برای مدیریت تاریخ‌ها (سال، ماه، روز) بدون در نظر گرفتن زمان.

- کلاس `time` برای مدیریت بخش‌های مختلف زمان مانند ساعت، دقیقه و ثانیه.

- کلاس `datetime` برای کار با ترکیب تاریخ و زمان، از جمله اطلاعات دقیق‌تر مانند ساعت و دقیقه.

### ۳. فرمتسازی و تبدیل تاریخ‌ها

- توانایی تبدیل تاریخ‌ها و زمان‌ها به فرمتهای مختلف مانند متن، عدد، و فرمتهای استاندارد

- مانند ISO 8601

- قابلیت مدیریت و قالب‌بندی تاریخ‌ها در قالب‌های دلخواه.

#### ۴. عملیات محاسباتی

- توانایی انجام عملیات ریاضی مانند جمع و تفریق تاریخ‌ها با استفاده از کلاس `timedelta` برای مدیریت فواصل زمانی.
  - انجام عملیات مربوط به محاسبه تفاوت‌های زمانی، محاسبه تاریخ آینده یا گذشته با توجه به فواصل زمانی.
۵. تایمزنونی و زمان‌های جهانی: پشتیبانی از مدیریت زمان‌های جهانی (UTC) و تبدیل بین زمان‌های مختلف با استفاده از کلاس `timezone` و `.tzinfo`
۶. پشتیبانی از محاسبات دقیق و معتبر: اطمینان از مدیریت دقیق مقادیر تاریخ و زمان با توجه به استانداردهای بین‌المللی، به ویژه برای سیستم‌های محاسباتی که نیاز به دقت بالا دارند.
۷. مدیریت تایم‌استمپ‌ها و لاغ‌گیری: قابلیت ذخیره‌سازی و مدیریت تاریخ و زمان رویدادها و ثبت واقعی در سیستم‌های لاغ‌گیری یا پایگاه داده‌ها.
۸. انعطاف‌پذیری و سازگاری: پشتیبانی از کار با تاریخ‌ها و زمان‌های مختلف، از جمله مقادیر معیوب یا `infinity` و `NaT` (Not a Time) نادرست مانند.
۹. تبدیل بین زمان محلی و زمان جهانی: امکان تبدیل تاریخ و زمان به زمان‌های محلی و بالعکس با استفاده از توابع خاص برای مدیریت مناطق زمانی مختلف.

این ویژگی‌ها موجب می‌شود که ماژول `datetime` یک ابزار بسیار مفید برای برنامه‌نویسان پایتون باشد که نیاز به مدیریت دقیق و انعطاف‌پذیر با تاریخ و زمان دارند.

## ۲-۱۵) کاربردهای مازول **datetime**

### ۱. مدیریت تاریخ و زمان

- استفاده برای ایجاد و مدیریت تاریخها و زمان‌ها در برنامه‌های مختلف.
- امکان کار با مقادیر مختلف مانند سال، ماه، روز، ساعت، دقیقه و ثانیه.

### ۲. محاسبات زمان و تاریخ

- انجام محاسبات زمان‌بندی مانند مقایسه دو تاریخ، پیدا کردن تفاوت زمانی، یا محاسبه فواصل زمانی بین دو تاریخ.
- انجام عملیات مربوط به افروzen یا کم کردن فواصل زمانی از یک تاریخ خاص.

### ۳. برنامه‌های مدیریتی و تقویمی

- استفاده در برنامه‌های مدیریت پروژه و سیستم‌های زمان‌بندی برای کنترل تاریخ‌های شروع و پایان کارها.
- پیاده‌سازی تقویم‌های شخصی، سازمانی یا سیستم‌های رزرو بلیط و هتل.

### ۴. تایم‌استمپ‌ها و ثبت وقایع

- ثبت و ذخیره‌سازی تاریخ‌ها و زمان‌ها برای رخدادها و وقایع مختلف در پایگاه داده‌ها.
- استفاده در سیستم‌های لاگ‌گیری برای ذخیره تغییرات و اقدامات مختلف.

### ۵. تبدیل فرمتهای مختلف تاریخ

- تبدیل تاریخ‌ها به فرمتهای مختلف مانند متن، عدد، یا قالب‌های استانداردی مانند ISO 8601

- استفاده از توابع برای تبدیل بین زمان محلی و جهانی.

#### ۶. تحلیل داده‌های زمانی

- تحلیل داده‌های آماری یا تاریخی که به اطلاعات دقیق تاریخ و زمان نیاز دارند.
- انجام عملیات داده‌کاوی و تحلیل روندهای زمانی مانند بررسی روند فروش، وضعیت آب و هوا یا داده‌های مالی.

#### ۷. محیط‌های برنامه‌نویسی وب

- استفاده از مژول `datetime` در برنامه‌های تحت وب برای پردازش درخواست‌های زمان‌دار، مدیریت جلسات کاربری یا اعتبارسنجی داده‌های زمانی.

#### ۸. اتوماسیون‌ها و مدیریت زمان

- استفاده در سیستم‌های خودکار که به محاسبات و تنظیمات بر اساس زمان نیاز دارند، مانند زمان‌بندی ارسال ایمیل‌ها، انجام عملیات خودکار یا اسکرپینگ داده‌ها.

#### ۹. برنامه‌های مالی و حسابداری

- استفاده از تاریخ‌ها برای مدیریت تراکنش‌ها، دوره‌های مالی، یا تاریخ‌های سرسید.
- انجام محاسبات سود یا ضرر با توجه به بازه‌های زمانی مختلف.

### ۳-۱۵) توابع وابسته‌ی موجود در مژول **Datetime**

ماژول `datetime` در پایتون شامل کlassen‌ها و توابع وابسته برای کار با تاریخ و زمان است. در جدول ۱-۱۵، توابع وابسته مهم این ماژول و توضیحات آن‌ها آمده است:

## جدول ۱۵-۱- جدول مربوط به تمامی توابع وابسته‌ی موجود در مژول *Datetime*

| توضیحات                                                                                                                           | تابع وابسته                           |
|-----------------------------------------------------------------------------------------------------------------------------------|---------------------------------------|
| تاریخ و زمان فعلی را بر اساس زمان سیستم برمی‌گرداند.                                                                              | <code>datetime.now()</code>           |
| تاریخ و زمان امروز را برمی‌گرداند (مشابه <code>now()</code> ، اما بدون تنظیم منطقه زمانی).                                        | <code>datetime.today()</code>         |
| یک رشته تاریخ/زمان را با استفاده از یک قالب مشخص تبدیل به یک شیء <code>datetime</code> می‌کند.                                    | <code>datetime.strptime()</code>      |
| شیء <code>datetime</code> را به یک رشته تاریخ/زمان با قالب مشخص تبدیل می‌کند.                                                     | <code>datetime.strftime()</code>      |
| تاریخ و زمان معادل یک زمان یونیکس (ثانیه از ۱ ژانویه ۱۹۷۰) را برمی‌گرداند.                                                        | <code>datetime.fromtimestamp()</code> |
| شیء <code>datetime</code> را به زمان یونیکس (ثانیه از ۱ ژانویه ۱۹۷۰) تبدیل می‌کند.                                                | <code>datetime.timestamp()</code>     |
| یک تاریخ از کلاس <code>date</code> و یک زمان از کلاس <code>time</code> را ترکیب کرده و یک شیء <code>datetime</code> ایجاد می‌کند. | <code>datetime.combine()</code>       |
| مقادیر مشخصی در یک شیء <code>datetime</code> را تغییر می‌دهد و شیء جدیدی ایجاد می‌کند.                                            | <code>datetime.replace()</code>       |
| زمان UTC معادل یک زمان یونیکس را برمی‌گرداند.                                                                                     | <code>datetime.utcnow()</code>        |
| شماره روز هفته را بر اساس استاندارد ISO برمی‌گرداند (۱=دوشنبه، ۷=یکشنبه).                                                         | <code>datetime.isoweekday()</code>    |
| شماره روز هفته را برمی‌گرداند (۰=دوشنبه، ۶=یکشنبه).                                                                               | <code>datetime.weekday()</code>       |
| یک رشته تاریخ/زمان در فرمت ISO 8601 تولید می‌کند.                                                                                 | <code>datetime.isoformat()</code>     |
| اطلاعات تاریخ و زمان را به صورت یک شیء <code>time.struct_time</code> (برای سازگاری با مژول <code>time</code> ) برمی‌گرداند.       | <code>datetime.timetuple()</code>     |
| شیء <code>datetime</code> را به یک منطقه زمانی خاص تبدیل می‌کند.                                                                  | <code>datetime.astimezone()</code>    |
| قسمت تاریخ از شیء <code>datetime</code> را به عنوان یک شیء <code>date</code> برمی‌گرداند.                                         | <code>datetime.date()</code>          |
| قسمت زمان از شیء <code>datetime</code> را به عنوان یک شیء <code>time</code> برمی‌گرداند.                                          | <code>datetime.time()</code>          |

|                                                                            |                                   |
|----------------------------------------------------------------------------|-----------------------------------|
| افست منطقه زمانی UTC مربوط به یک شیء <code>datetime</code> را برمی‌گرداند. | <code>datetime.utcnow()</code>    |
| تغییرات ساعت تابستانی <sup>۹۹</sup> را برمی‌گرداند (در صورت وجود).         | <code>datetime.dst()</code>       |
| سال را به صورت یک عدد صحیح برمی‌گرداند.                                    | <code>datetime.year</code>        |
| ماه را به صورت یک عدد صحیح (۱ تا ۱۲) برمی‌گرداند.                          | <code>datetime.month</code>       |
| روز ماه را به صورت یک عدد صحیح برمی‌گرداند.                                | <code>datetime.day</code>         |
| ساعت را به صورت یک عدد صحیح (۰ تا ۲۳) برمی‌گرداند.                         | <code>datetime.hour</code>        |
| دقیقه را به صورت یک عدد صحیح (۰ تا ۵۹) برمی‌گرداند.                        | <code>datetime.minute</code>      |
| ثانیه را به صورت یک عدد صحیح (۰ تا ۵۹) برمی‌گرداند.                        | <code>datetime.second</code>      |
| میکروثانیه را به صورت یک عدد صحیح (۰ تا ۹۹۹۹۹۹) برمی‌گرداند.               | <code>datetime.microsecond</code> |

## ۴-۱۵) توابع مهم و پرکاربرد مازول **Datetime**

### ۱-۴-۱۵) برگرداندن تاریخ و زمان فعلی سیستم

python example - `datetime_module.py`

```

1 # datetime module in Python
2 from datetime import datetime # وارد کردن مازول و کلاس
3
4 # datetime.now()
5 now = datetime.now()
6 print(now)

```

شکل ۱-۱۵-۱- مالی از گرفتن تاریخ و زمان فعلی سیستم با استفاده از مازول `datetime`

خروجی شکل ۱-۱۵:

<sup>۹۹</sup> Daylight Saving Time

---

2025-01-08 11:20:18.605510

---

#### ۲-۴-۱۵) برگرداندن تاریخ و زمان فعلی سیستم بدون تنظیمات منطقه زمانی

python example - datetime\_module.py

```
10 # datetime module in Python
11 from datetime import datetime # وارد کردن مازول و کلاس
12
13 # datetime.today()
14 today = datetime.today()
15 print(today)
```

شکل ۲-۱۵-۲- مثالی از گرفتن تاریخ و زمان فعلی سیستم بدون تنظیمات منطقه زمانی با استفاده از مازول *datetime*

خروجی شکل ۲-۱۵ :

---

2025-01-08 11:25:01.504428

---

## ۳-۴-۱۵ تبدیل یک رشته تاریخ/زمان به یک شیء **datetime**

python example - datetime\_module.py

```
19 # datetime module in Python
20 from datetime import datetime # وارد کردن ماثول و کلاس
21
22 تبدیل یک رشته تاریخ/زمان به یک شیء #
23 date_str = "2024-12-29 15:45"
24 date_obj = datetime.strptime(date_str, "%Y-%m-%d %H:%M")
25
26 print(date_obj)
27 print(f"Type of date_str is {type(date_str)}")
28 print(f"Type of date_obj is {type(date_obj)}")
```

شكل ۳-۱۵-۳- مثالی از تبدیل یک رشته تاریخ/زمان به شیء **datetime**

خروجی شکل ۳-۱۵:

---

2024-12-29 15:45:00

Type of date\_str is <class 'str'>

Type of date\_obj is <class 'datetime.datetime'>

---

#### ۴-۴-۱۵) تبدیل یک شیء `datetime` را به رشته تاریخ/زمان

python example - datetime\_module.py

```
33 # datetime module in Python
34 from datetime import datetime # وارد کردن ماثول و کلاس
35
36 now = datetime.now()
37 formatted = now.strftime("%Y/%m/%d %H:%M:%S")
38 # Y: year, m: month, d: day, H: hour, M: minute, S: second
39
40 print(formatted)
41 print(f"type of now is: {type(now)}")
42 print(f"type of formatted is: {type(formatted)}")
```

شكل ۴-۱۵-۴- مثالی از تبدیل یک شیء `datetime` به رشته تاریخ/زمان

خروجی شکل ۴-۱۵

---

2025/01/08 11:39:58

*type of now is: <class 'datetime.datetime'>*

*type of formatted is: <class 'str'>*

---

#### ۴-۴-۱۶) برگرداندن تاریخ و زمان معادل یک زمان یونیکس

نکته: تعداد ثانیه‌ها از ۱ ژانویه ۱۹۷۰ محاسبه می‌شود.

python example - datetime\_module.py

```
46 # datetime module in Python
47 from datetime import datetime # وارد کردن ماثول و کلاس
48
49 timestamp = 1937210700
50 date_from_timestamp = datetime.fromtimestamp(timestamp)
51 print(date_from_timestamp)
```

شکل ۱۵-۵-مثالی از برگرداندن تاریخ و زمان معادل یک زمان یونیکس با استفاده از ماثول *datetime*

خروجی شکل ۱۵-۵:

---

2031-05-22 13:35:00

---

۱۵-۴-۶) ترکیب یک تاریخ و زمان و ایجاد یک شیء **datetime**

python example - datetime\_module.py

```
55 # datetime module in Python
56 from datetime import datetime, date, time # وارد کردن ماثول و کلاس
57
58 d = date(2025, 1, 8)
59 t = time(11, 35, 40)
60 combined = datetime.combine(d, t)
61 print(combined)
```

شکل ۱۵-۶-مثالی از ترکیب یک تاریخ و زمان و ایجاد یک شیء *datetime*

خروجی شکل ۱۵-۶:

---

2025-01-08 11:35:40

---

## ۷-۴-۱۵) تغییر دادن قسمتی از یک شی **datetime**

python example - datetime\_module.py

```
65 # datetime module in Python مازول تاریخ و زمان در پایتون
66 from datetime import datetime # وارد کردن مازول و کلاس
67
68 now = datetime.now() تغییر دادن ساعت و دقیقه
69 print(f"Before Changing: {now}")
70 new_time = now.replace(hour=20, minute=30) # مثالی از تغییر دادن قسمتی از یک شی
71 print(f"After Changing: {new_time}")
```

شکل ۷-۱۵-۷-۱۵) مثالی از تغییر دادن قسمتی از یک شی *datetime*

خروجی شکل ۷-۱۵

---

*Before Changing:* 2025-01-08 11:52:41.586638

*After Changing:* 2025-01-08 20:30:41.586638

---

## ۸-۴-۱۵) برگرداندن زمان یونیکس معادل یک شیء **datetime**

python example - datetime\_module.py

```
75 # datetime module in Python
76 from datetime import datetime # وارد کردن مازول و کلاس
77
78 now = datetime.now()
79 print(f"Date Time: {now}")
80 timestamp = now.timestamp()
81 print(f"Unix Time: {timestamp}")
```

شکل ۸-۱۵-۸-مثالی از برگرداندن زمان یونیکس معادل یک شیء **datetime**

خروجی شکل ۸-۱۵

---

*Date Time: 2025-01-08 11:55:13.229226*

*Unix Time: 1736324713.229226*

---

# فصل شانزدهم

آشنایی با مازول نامپای در پایتون

## ۱۶) آشنایی با ماژول نامپای در پایتون

ماژول نامپای<sup>۱۰۰</sup> یک کتابخانه محبوب و بسیار کارآمد در پایتون است که برای کار با داده‌های عددی و انجام محاسبات علمی طراحی شده است. این ماژول به دلیل ارائه آرایه‌های چندبعدی<sup>۱۰۱</sup> و مجموعه‌ای از توابع ریاضی، آماری و جبری، یک ابزار کلیدی برای دانشمندان داده، مهندسان، و تحلیل‌گران است.

### ۱-۱۶) ویژگی‌های اصلی نامپای

۱. آرایه‌های چندبعدی: هسته نامپای بر پایه آرایه‌های چندبعدی بنا شده است. آرایه‌های نامپای از

فهرست‌های پایتون سریع‌تر و کارآمدتر هستند زیرا:

◦ از یک ساختار داده‌ای فشرده استفاده می‌کنند.

◦ عملیات‌های ریاضی بر روی آرایه‌ها بدون نیاز به حلقه‌های پایتون انجام می‌شوند.

◦ بهینه‌سازی شده برای پردازش برداری<sup>۱۰۲</sup>.

۲. کار با داده‌های بزرگ: نامپای به دلیل مصرف بهینه حافظه و اجرای سریع محاسبات، برای پردازش

داده‌های حجمی بسیار مناسب است.

۳. توابع ریاضی و آماری پیشرفته: نامپای دارای مجموعه‌ای از توابع داخلی برای انجام محاسباتی مانند

میانگین، واریانس، انحراف معیار، انتگرال‌گیری، و بسیاری دیگر است.

<sup>100</sup> NumPy

<sup>101</sup> ndarray

<sup>102</sup> Vectorized Operations

۴. عملیات ماتریسی و جبر خطی: نامپای ابزارهای گسترهای برای انجام عملیات ماتریسی مانند ضرب ماتریس‌ها، محاسبه دترمینان، مقادیر ویژه، و حل معادلات خطی فراهم می‌کند.

۵. سازگاری بالا با سایر کتابخانه‌ها: نامپای به عنوان پایه‌ای برای بسیاری از کتابخانه‌های دیگر در اکوسیستم پایتون (مانند TensorFlow، SciPy، Pandas) عمل می‌کند.

۶. ساخت داده‌های خاص: نامپای ابزارهایی برای ساخت آرایه‌های عددی خاص مانند آرایه‌های صفر، آرایه‌های یک، دنباله‌های عددی خطی یا آرایه‌های تصادفی فراهم می‌کند.

## ۲-۱۶) کاربردهای نامپای

- ❖ تحلیل داده: پردازش، پاکسازی و تحلیل مجموعه‌های داده.
- ❖ محاسبات علمی: انجام شبیه‌سازی‌ها، جبر خطی و مدل‌سازی ریاضی.
- ❖ یادگیری ماشین: آماده‌سازی داده‌ها و محاسبات برداری برای الگوریتم‌های یادگیری ماشین.
- ❖ گرافیک کامپیوتری: کار با داده‌های تصویری و ویدئویی.
- ❖ آمار: تحلیل و محاسبه معیارهای آماری داده‌ها.

## ۳-۱۶) مزایای نامپای

- ✓ سرعت بالا: اجرای سریع‌تر محاسبات عددی نسبت به ساختارهای پیش‌فرض پایتون.
- ✓ انعطاف‌پذیری: ابزارهایی برای انجام طیف گسترده‌ای از عملیات بر روی داده‌ها.
- ✓ قابلیت مقیاس‌پذیری: امکان کار با داده‌های بسیار بزرگ بدون کاهش عملکرد.

## ۴-۱۶) مقایسه با فهرست‌ها در پایتون

- آرایه‌های نامپای به دلیل نوع داده یکنواخت و عملیات برداری، سریع‌تر و کارآمدتر از فهرست‌های پایتون هستند.
- نامپای از ابزارهایی برای تغییر شکل، برش و انجام محاسبات ریاضی برخوردار است که به سادگی با فهرست‌های پایتون امکان‌پذیر نیست.

به طور کلی، نامپای یکی از ابزارهای کلیدی برای محاسبات عددی و کار با داده‌های علمی است. این مژول به دلیل سرعت بالا، تنوع ابزارها، و قابلیت سازگاری با سایر کتابخانه‌ها، تقریباً در تمام پژوهش‌های علمی و داده‌محور مورد استفاده قرار می‌گیرد.

## ۵-۱۶) نصب نامپای

برای نصب نامپای، می‌توانید از ابزار مدیریت بسته پایتون یعنی **pip** استفاده کنید. در ادامه مراحل نصب شرح داده شده است:

### ۱-۵-۱۶) نصب مژول نامپای با **pip**

در خط فرمان (Terminal یا Command Prompt) دستور زیر را بنویسید و اجرا کنید:

---

***pip install numpy***

---

### ۲-۵-۱۶) بررسی نصب مژول نامپای

پس از نصب، برای اطمینان از نصب صحیح، دستور زیر را در ترمینال اجرا کنید:

---

```
python -c "import numpy; print(numpy.__version__)"
```

---

اگر شماره نسخه نمایش داده شد، نصب موفقیت‌آمیز بوده است.

### ۳-۵-۱۶) نصب نسخه خاص از ماژول نامپای

برای نصب یک نسخه مشخص از نامپای:

---

```
pip install numpy==1.22.0
```

---

### ۴-۵-۱۶) بروزرسانی ماژول نامپای

برای بروزرسانی به آخرین نسخه:

---

```
pip install --upgrade numpy
```

---

### ۵-۵-۱۶) نصب ماژول نامپای در محیط مجازی

اگر در یک محیط مجازی<sup>۱۰۳</sup> کار می‌کنید، ابتدا محیط مجازی را فعال کرده و سپس نامپای را نصب کنید:

---

```
pip install numpy
```

---

---

<sup>103</sup> Virtual Environment

## ۶-۱۶) ماتریس‌ها در نامپای

ماتریس‌ها در نامپای یکی از پرکاربردترین ابزارها برای محاسبات ریاضی، جبر خطی، و تحلیل داده‌ها هستند. در واقع، ماتریس‌ها در نامپای به عنوان یک نوع خاص از آرایه‌های دوبعدی در نظر گرفته می‌شوند.

### ۱-۶-۱۶) ویژگی‌های ماتریس‌ها در نامپای

۱. ساختار دوبعدی: ماتریس‌ها همیشه دارای دو بعد هستند (سطر و ستون).
۲. عملیات برداری و اسکالر: تمام عملیات ریاضی مانند جمع، ضرب، و توان به صورت عنصر به عنصر یا برداری روی ماتریس‌ها انجام می‌شود.
۳. توابع جبر خطی: نامپای شامل توابعی برای محاسبات ماتریسی پیشرفته مانند دترمینان، معکوس، ضرب ماتریسی، و مقادیر ویژه است.
۴. تفاوت با ماتریس: آرایه‌های دوبعدی برای ماتریس‌ها توصیه می‌شوند، زیرا انعطاف بیشتری دارند. نوع داده‌ی ماتریس به‌طور خاص در نامپای وجود دارد اما کمتر استفاده می‌شود.

### ۲-۶-۱۶) ایجاد ماتریس در ماژول نامپای

برای ایجاد ماتریس‌ها، از آرایه‌های دوبعدی استفاده می‌شود. در ادامه به روش‌های ایجاد ماتریس در پایتون پرداخته می‌شود.

### ۳-۶-۱۶) با استفاده از فهرست‌های تو در تو

---

*import numpy as np*

```
mat = np.array([[1, 2], [3, 4]])
```

---

۱۶-۶-۲-۲) ایجاد ماتریس‌های خاص:

۱۶-۶-۲-۲-۱) ماتریس صفر

python example - numpy\_module.py

```
1 # Numpy in Python نامپای در پایتون
2 # Install numpy: نصب نامپای
3 #     pip install numpy
4
5 import numpy as np
6
7 # یجاد ماتریس صفر ۳*۳
8 print(np.zeros((3,3)))
```

شکل ۱۶-۱- مثال مربوط به ایجاد ماتریس صفر  $3 \times 3$  با استفاده از ماژول نامپای

خروجی شکل ۱۶-۱:

---

$[[0, 0, 0],$

$[0, 0, 0],$

$[0, 0, 0]]$

---

python example - numpy\_module.py

```
12 # Numpy in Python نامپای در پایتون
13 # Install numpy: نصب نامپای
14 #     pip install numpy
15
16 import numpy as np
17
18 # # 3*3 ایجاد ماتریس یک
19 print(np.ones((3,3)))
```

شکل ۱۶-۲-مثال مربوط به ایجاد ماتریس  $3 \times 3$  با درایه‌های یک با استفاده از ماثول نامپای

خروجی شکل ۱۶-۲:

---

[[1, 1, 1],

[1, 1, 1],

[1, 1, 1]]

---

### ۳-۲-۶-۱۶) ماتریس واحد (قطری)

python example - numpy\_module.py

```
23 # Numpy in Python نامپای در پایتون
24 # Install numpy: نصب نامپای
25 #       pip install numpy
26
27 import numpy as np
28
29 # 3*3 ایجاد ماتریس قطری
30 print(np.eye(3))
```

شکل ۳-۱۶-۳- مثال مربوط به ایجاد ماتریس قطری  $3 \times 3$  با استفاده از ماژول نامپای

خروجی شکل ۳-۱۶:

---

[1. 0. 0.]

[0. 1. 0.]

[0. 0. 1.]]

---

### ۴-۲-۶-۱۶) آرایه‌های چندبعدی

در ماژول نامپای، تابع `np.array()` برای ایجاد آرایه‌های چندبعدی استفاده می‌شود. این آرایه‌ها ساختاری مشابه لیست‌ها در پایتون دارند اما با قابلیت‌های بسیار بیشتر برای پردازش داده‌های عددی.

python example - numpy\_module.py

```
34 # Numpy in Python نامپای در پایتون
35 # Install numpy: نصب نامپای
36 #     pip install numpy
37
38 import numpy as np
39
40 # ایجاد ماتریس 3*3
41 print(np.array([[1, 2, 3], [4, 5, 6]]))
```

شکل ۴-۱۶-۴-مثال مربوط به ایجاد آرایه‌های چندبعدی در نامپای

خروجی شکل ۴-۱۶:

---

$\begin{bmatrix} 1 & 2 & 3 \end{bmatrix}$

---

$\begin{bmatrix} 4 & 5 & 6 \end{bmatrix}$

---

۳-۶-۱۶) عملیات‌های اصلی ماتریسی

۳-۶-۱۶) جمع و تفریق ماتریس‌ها

انجام عملیات جمع و تفریق به صورت عنصر به عنصر:

### python example - numpy\_module.py

```
45 # Numpy in Python نامپای در پایتون
46
47 import numpy as np
48 A = np.array([[1, 2, 3], [4, 5, 6]])
49 B = np.array([[10, 20, 30], [40, 50, 60]])
50
51 # عملیات جمع در ماتریس ها
52 print(f"A + B = \n{A+B}")
53
54 # عملیات تفریق در ماتریس ها
55 print(f"A - B = \n{A-B}")
```

شکل ۱۶-۵- مثالی از عملیات جمع بین ماتریس‌ها در مازول نامپای

خروجی شکل ۱۶-۵:

---

$$A + B =$$

$$[[11\ 22\ 33]]$$

$$[44\ 55\ 66]]$$

$$A - B =$$

$$[[ -9\ -18\ -27]]$$

$$[-36\ -45\ -54]]$$

---

۱۶-۳-۲) ضرب ماتریسی

برای ضرب ماتریسی، از تابع وابسته `np.dot()` یا عملگر `@` استفاده می‌شود:

python example - numpy\_module.py

```
59 # Numpy in Python
60
61 import numpy as np
62
63 a = np.array([[1, 0], [0, 1]])
64 b = np.array([[4, 1], [2, 2]])
65
66 # np.dot()
67 print(np.dot(a, b))
```

شکل ۱۶-۶- مثالی از ضرب ماتریسی در ماژول نامپای

خروجی شکل ۱۶-۶:

---

[1 4 1]

[2 2]

---

۱۰۴) ترانهاده ۳-۶-۱۶

جابجایی عناصر قطری یک ماتریس:

---

<sup>104</sup> Transpose

python example - numpy\_module.py

```
71 # Numpy in Python
72
73 import numpy as np
74
75 a = np.array([[4, 1], [2, 2]])
76 print(f"Before: \n{a}")
77
78 # transpose()
79 print(f"After (transpose()): \n{np.transpose(a)}")
80
81 # a.T
82 print(f"After (a.T): \n{a.T}")
```

شکل ۱۶-۷- مثالی از جابجایی عناصر قطری یک ماتریس در ماژول نامپای

خروجی شکل ۱۶-۷:

---

*Before:*

$\begin{bmatrix} 4 & 1 \\ 2 & 2 \end{bmatrix}$

$\begin{bmatrix} 2 & 2 \\ 1 & 4 \end{bmatrix}$

*After (transpose()):*

$\begin{bmatrix} 4 & 2 \\ 1 & 2 \end{bmatrix}$

$\begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix}$

*After (a.T):*

$\begin{bmatrix} 4 & 2 \\ 1 & 2 \end{bmatrix}$

$\begin{bmatrix} 2 & 1 \\ 4 & 2 \end{bmatrix}$

## ۱۶-۶-۴) معکوس ماتریس

محاسبه معکوس با تابع `:np.linalg.inv()`

python example - numpy\_module.py

```
86 # Numpy in Python نامپای در پایتون
87
88 import numpy as np
89 a = np.array([[4, 1], [2, 2]])
90 print(f"Default Matrix: \n{a}")
91
92 # معکوس ماتریس
93 print(f"Invertible matrix: \n{np.linalg.inv(a)}")
```

شکل ۱۶-۸-مثال مربوط به محاسبه معکوس ماتریس در ماژول نامپای

خروجی شکل ۸-۱۶:

---

*Default Matrix:*

$\begin{bmatrix} 4 & 1 \\ 2 & 2 \end{bmatrix}$

*Invertible matrix:*

$\begin{bmatrix} 0.33333333 & -0.16666667 \\ -0.33333333 & 0.66666667 \end{bmatrix}$

## ۱۶-۶-۵) دترمینان ماتریس

محاسبه دترمینان با `:np.linalg.det()`

python example - numpy\_module.py

```
97 # Numpy in Python
98
99 import numpy as np
100 a = np.array([[4, 1], [2, 2]])
101 print(f"Default Matrix: \n{a}")
102
103 # دترمینال ماتریس
104 print(f"Matrix Determinant: \n{np.linalg.det(a)}")
```

شکل ۱۶-۹-مثال مربوط به محاسبه دترمینال ماتریس در ماژول نامپای

خروجی شکل ۹-۱۶:

---

*Default Matrix:*

*[[4 1]*

*[2 2]]*

*Matrix Determinant:*

*6.0*

---

## ۱۶-۷) دسترسی به عناصر ماتریس

## ۱۶-۶-۷) دسترسی به عنصر خاص

python example - numpy\_module.py

```
108 # Numpy in Python نامپای در پایتون
109
110 import numpy as np
111 a = np.array([[4, 1], [2, 5]])
112 print(f"Default Matrix: \n{a}")
113
114 دسترسی به عناصر ماتریس # دسترسی به عنصر خاص از ماتریس
115 # دسترسی به عنصر خاص از ماتریس
116 print(f'a[0, 0]: {a[0,0]}')
117 print(f'a[1, 1]: {a[1,1]}')
118 print(f'a[1, 0]: {a[1,0]}')
119 print(f'a[0, 1]: {a[0,1]}')
```

شکل ۱۶-۱۰- مثالی از دسترسی به عنصر خاصی از ماتریس

خروجی شکل ۱۰-۱۶:

### ***Default Matrix:***

[4 1]

[2 5]]

*a[0, 0]: 4*

*a[1, 1]; 5*

*a[1, 0]; 2*

*a[0, 1]; 1*

## ۱۶-۶-۷-۲) دسترسی به یک سطر یا ستون

python example - numpy\_module.py

```
123 # Numpy in Python
124
125 import numpy as np
126 a = np.array([[4, 1], [2, 5]])
127 print(f"Default Matrix: \n{a}")
128
129 دسترسی به عناصر ماتریس #
130 دسترسی به سطر یا ستون یک ماتریس #
131 سطر اول #
132 print(f"a[0, :]: {a[0, :]})"
133 سطر دوم
134 print(f"a[1, :]: {a[1, :]})"
135 ستون اول #
136 print(f"a[:, 0]: {a[:, 0]}"))
137 ستون دوم #
138 print(f"a[:, 1]: {a[:, 1]}")
```

شکل ۱۱-۱۶-مثالی از مثالی از دسترسی به سطر یا ستون مورد نظر از ماتریس

خروجی شکل ۱۱-۱۶:

---

*Default Matrix:*

*[[4 1]*

*[2 5]]*

*a[0, :]: [4 1]*

*a[1, :]: [2 5]*

*a[:, 0]: [4 2]*

$a[:, 1]: [1 \ 5]$

### ١٦-٨) تغییر شکل ماتریس:

#### ۱۶-۸-۱) تغییر شکل ماتریس با reshape

python example - numpy\_module.py

```
142 # Numpy in Python نامپای در پایتون
143
144 import numpy as np
145 a = np.array([[4, 1, 9], [2, 5, 2]])
146 print(f"Default Matrix: \n{a}")
147
148 # تغییر شکل ماتریس
149
150 print(f"Reshape a: \n{a.reshape((3, 2))}") # تغییر شکل ماتریس به ۳ سطر و ۲ ستون
```

### شکل ۱۶-۱۲- مثال مربوط به تغییر شکل ماتریس

خروجی شکل ۱۶-۱۲:

### ***Default Matrix:***

[419]

[2 5 2]]

*Reshape a:*

[4 1]

[9 2]

نکته: مقایسه ماتریس و آرایه‌ی چند بعدی

- ✓ آرایه‌ی چند بعدی: انعطاف بیشتری دارد و برای اکثر کارها توصیه می‌شود.
- ✓ ماتریس: نوعی خاص از آرایه‌ی چند بعدی است که فقط برای ماتریس‌ها طراحی شده اما استفاده از آن منسوخ شده است.

## ۷-۱۶) مثال‌های کاربردی نامپای

### ۱-۷-۱۶) ایجاد آرایه

مثال: با استفاده از مازول نامپای آرایه‌های یکبعدی و دوبعدی ایجاد کرده و در خروجی چاپ کنید.

python example - numpy\_module.py

```

154 # Numpy in Python
155
156 # Example 1. Create 1d and 2d array
157 import numpy as np
158
159 # آرایه یکبعدی
160 array_1d = np.array([1, 2, 3, 4])
161 print(f"1D array: \n{array_1d}")
162 # آرایه دو بعدی
163 array_2d = np.array([[1, 2], [3, 4]])
164 print(f"2D array: \n{array_2d}")

```

شکل ۱۳-۱۶- مثال مربوط به ایجاد آرایه یکبعدی و دو بعدی با استفاده از مازول نامپای

---

*1D array:*

`[1 2 3 4]`

*2D array:*

`[[1 2]`

`[3 4]]`

---

### ۱۶-۷-۳) ایجاد آرایه‌های خاص

مثال: با استفاده از ماثول نامپای، آرایه‌ای از صفر، یک، اعداد متوالی و آرایه‌ای از اعداد بین دو مقدار ایجاد کرده و در خروجی چاپ کنید.

python example - numpy\_module.py

```
168 # Numpy in Python نامپای در پایتون
169
170 # Example 2. Create Specia array
171 import numpy as np
172
173 # آرایه‌ای از صفرها
174 zeros = np.zeros((2, 3))
175 print(f"Zeros Array: \n{zeros}")
176
177 # آرایه‌ای از یکها
178 ones = np.ones((3, 3))
179 print(f"Ones Array: \n{ones}")
180
181 # آرایه‌ای از اعداد متوالی
182 sequence = np.arange(1, 10, 2)
183 print(f"Sequence 2D array: \n{sequence}")
184
185 # آرایه‌ای از اعداد بین دو مقدار
186 linspace = np.linspace(0, 1, 5)
187 print(f"Numbric Array between two number:\n{linspace}")
```

شکل ۱۶-۱۶- مثال مربوط به ایجاد آرایه‌های خاص

خروجی شکل ۱۶-۱۶:

---

*Zeros Array:*

*[[0. 0. 0.]*

*[0. 0. 0.]]*

*Ones Array:*

*[[1. 1. 1.]*

*[1. 1. 1.]]*

*[1. 1. 1.]*

*Sequence 2D array:*

*[1 3 5 7 9]*

*Numbric Array between two number:*

*[0. 0.25 0.5 0.75 1.]*

---

### ۱۶-۷-۳) شکل و اندازه آرایه

مثال: با استفاده از مازول نامپای، اندازه، شکل و بعد یک آرایه را بدست آورید.

python example - numpy\_module.py

```
191 # Numpy in Python
192
193 # Example 3. Get array shape, size and dimension
194 import numpy as np
195
196 array = np.array([[1, 2, 3], [4, 5, 6]])
197 print(f"Shape: {array.shape}")
198 print(f"Size: {array.size}")
199 print(f"Dimension: {array.ndim}")
```

شکل ۱۵-۱۶-مثالی از به دست آوردن اندازه، شکل و بعد یک آرایه با استفاده از مازول نامپای

: ۱۶-۱۵ خروجی شکل

---

*Shape: (2, 3)*

*Size: 6*

## Dimension: 2

---

### ۱۶-۷-۴) عملیات عددی

مثال: با استفاده از ماثول نامپای، عملیات جمع و تفریق، ضرب و تقسیم و توان دوم هر آرایه را بدست آورید.

python example - numpy\_module.py

```
203 # Numpy in Python نامپای در پایتون
204
205 # Example 4. obtain the addition, subtraction, multiplication, division,
206 # and square root operations of any array.
207
208 import numpy as np
209 a = np.array([1, 2, 3])
210 b = np.array([4, 5, 6])
211
212 # آرایه ها
213 print(f"a = {a}")
214 print(f"b = {b}")
215
216 # جمع و تفریق
217 print(f"a + b = {a + b}")
218 print(f"a - b = {a - b}")
219
220 # ضرب و تقسیم
221 print(f"a * b = {a * b}")
222 print(f"a / b = {a / b}")
223
224 # توان
225 print(f"a**2 = {a**2}")
226 print(f"b**2 = {b**2}")
```

شکل ۱۶-۱۶-۱۶- مثال مربوط به عملیات جمع و تفریق، ضرب و تقسیم و توان دوم برای دو آرایه

## خروجی شکل ۱۶-۱۶:

---

$$a = [1 \ 2 \ 3]$$

$$b = [4 \ 5 \ 6]$$

$$a + b = [5 \ 7 \ 9]$$

$$a - b = [-3 \ -3 \ -3]$$

$$a * b = [4 \ 10 \ 18]$$

$$a / b = [0.25 \ 0.4 \ 0.5]$$

$$a^{**2} = [1 \ 4 \ 9]$$

$$b^{**2} = [16 \ 25 \ 36]$$

---

## ۱۶-۷) توابع ریاضی

مثال: با استفاده از ماژول نامپای، یک آرایه یکبعدی ایجاد کرده و سپس تابع ریاضی مانند جمع، میانگین، ماکزیمم، مینیمم و انحراف استاندارد را روی آرایه پیاده‌سازی کنید.

### python example - numpy\_module.py

```
230 # Numpy in Python
231
232 # Example 5. obtain the addition, avrage, maximum, minimum,
233 #           and Standard deviation of 1D array.
234
235 import numpy as np
236
237 array = np.array([1, 2, 3, 4])
238 print(f"array: {array}")
239 print(f"Sum of array: {np.sum(array)}")
240 print(f"Average of array: {np.mean(array)}")
241 print(f"Max of array: {np.max(array)}")
242 print(f"Min of array: {np.min(array)}")
243 print(f"Standard deviation of array: {np.std(array)}")
```

شکل ۱۶-۱۷- مثالی از اعمال برخی اعمال ریاضی روی درایه‌های یک آرایه

خروجی:

---

*array: [1 2 3 4]*

*Sum of array: 10*

*Average of array: 2.5*

*Max of array: 4*

*Min of array: 1*

---

# فصل هفدهم

آشنایی با ماژول SymPy در پایتون

## ۱۷) آشنایی با ماژول SymPy در پایتون

ماژول SymPy یک کتابخانهٔ نمادین<sup>۱۰۵</sup> در پایتون است که برای انجام محاسبات ریاضی نمادین<sup>۱۰۶</sup> طراحی شده است. SymPy یک کتابخانهٔ متن‌باز در زبان برنامه‌نویسی پایتون است که برای انجام محاسبات ریاضی نمادین طراحی شده است. این کتابخانه به عنوان یک سیستم جبری کامپیوتری<sup>۱۰۷</sup> عمل می‌کند و به کاربران امکان می‌دهد تا مسائل ریاضی را به صورت نمادین (و نه عددی) تحلیل و حل کنند. این ماژول ابزارهایی برای حل معادلات، انجام محاسبات جبری، محاسبه انتگرال و مشتق، ساده‌سازی عبارات ریاضی، و بسیاری موارد دیگر فراهم می‌کند.

## ۱-۱۷) ویژگی‌های SymPy

۱. محاسبات ریاضی نمادین: SymPy به کاربران امکان می‌دهد تا عبارات ریاضی را به صورت نمادین مدیریت کنند. این ویژگی روی شکل جبری و نمادین معادلات و توابع مرکز دارد. برخلاف کتابخانه‌هایی مانند نامپای که محاسبات عددی انجام می‌دهند، SymPy می‌تواند عبارات ریاضی را به صورت نمادین مانند  $y + xy$  مدیریت کند.

۲. کتابخانه‌ای سبک و قابل حمل: SymPy کاملاً در پایتون نوشته شده است و به هیچ وابستگی خارجی نیاز ندارد. این موضوع باعث می‌شود که نصب و استفاده از آن آسان باشد.

۳. متن‌باز و قابل گسترش: SymPy یک پروژهٔ متن‌باز است که تحت مجوز BSD عرضه می‌شود. این مجوز به توسعه‌دهندگان اجازه می‌دهد تا به راحتی کد را بررسی، تغییر یا گسترش دهند. SymPy امکان

<sup>105</sup> Symbolic

<sup>106</sup> Symbolic Mathematics

<sup>107</sup> CAS: Computer Algebra System

انجام عملیات جبری مانند سادهسازی، فاکتورگیری، گسترش عبارات، و محاسبه چندجمله‌ای‌ها را فراهم می‌کند.

۴. سادگی و انعطاف‌پذیری: SymPy طوری طراحی شده است که حتی برای کاربران تازه‌کار نیز قابل استفاده باشد. با این حال، ابزارها و قابلیت‌های پیچیده‌ای را برای تحلیل مسائل پیشرفته ریاضی فراهم می‌کند.

۵. سازگاری با سایر ابزارها: SymPy می‌تواند با سایر کتابخانه‌ها و ابزارهای علمی مانند SciPy، NumPy و Matplotlib ترکیب شود. همچنین قابلیت تبدیل عبارات ریاضی به زبان‌های برنامه‌نویسی دیگر مانند JavaScript، Fortran، C

۶. تبدیل به کد: SymPy می‌تواند عبارات ریاضی را به زبان‌های برنامه‌نویسی مانند سی، فورترن یا جاوااسکریپت تبدیل کند.

۷. قابلیت گسترده در جبر ریاضی: SymPy امکان انجام عملیات جبری مانند سادهسازی، فاکتورگیری، گسترش عبارات، و محاسبه چندجمله‌ای‌ها را فراهم می‌کند.

۸. محاسبات دیفرانسیل و انتگرال: مشتق و انتگرال‌گیری از توابع به صورت دقیق و نمادین.

۹. حل معادلات: حل معادلات جبری، دیفرانسیلی و سیستم‌های معادلات.

## ۲-۱۷) کاربردهای SymPy

۱. تحلیل جبری: سادهسازی، فاکتورگیری، گسترش عبارات ریاضی، و تحلیل چندجمله‌ای‌ها.

۲. حل معادلات: حل معادلات جبری، سیستم‌های معادلات، و معادلات دیفرانسیل.

۳. محاسبات دیفرانسیلی و انتگرالی: محاسبه مشتقات و انتگرال‌های دقیق به صورت نمادین.
۴. آنالیز عددی و نمادین: تحلیل دقیق عبارات و مقادیر ریاضی با استفاده از ابزارهای پیشرفته.
۵. تولید کد: تولید کد به زبان‌های مختلف برای استفاده در برنامه‌های دیگر.
۶. آموزش ریاضیات: برای دانشجویان و محققانی که می‌خواهند مفاهیم ریاضی را بهتر درک کنند.
۷. تحقیقات علمی: حل مسائل پیچیده ریاضی و تحلیل نمادین.
۸. مهندسی و فیزیک: حل معادلات دیفرانسیل و انجام تحلیل سیستم‌های پیچیده.

## ۳-۱۷) نقاط قوت SymPy

۱. کاملاً مبتنی بر پایتون: به دلیل اینکه SymPy به طور کامل در پایتون نوشته شده است، کاربران پایتون به راحتی می‌توانند از آن استفاده کنند.
۲. مستندات جامع: SymPy دارای مستندات کاملی است که کاربران را در یادگیری و استفاده از ابزارها و قابلیت‌های مختلف راهنمایی می‌کند.
۳. پشتیبانی از موضوعات متنوع ریاضی: شامل جبر خطی، محاسبات جبری، حل معادلات دیفرانسیل، سری‌های ریاضی، و حتی هندسه تحلیلی.
۴. انعطاف‌پذیری بالا: SymPy می‌تواند در پروژه‌های مختلف از آموزش ریاضیات تا تحقیقات علمی و مهندسی استفاده شود.
۵. رابط گرافیکی: SymPy می‌تواند با رابطهای گرافیکی مانند Jupyter Notebook ترکیب شود و تحلیل‌های ریاضی را به صورت تعاملی و قابل درک نمایش دهد.

۶. حل ساده سیستم‌های پیچیده: سبک‌تر و ساده‌تر از برخی سیستم‌های پیچیده‌تر جبری است.

۷. برتری نسبت به سیستم‌های مشابه: نسبت به سیستم‌های مشابه مانند Maple و Mathematica یا

RaiGAN و SymPy متن‌باز است.

## ۴-۱۷) نصب و وارد کردن ماژول SymPy

برای نصب ماژول SymPy از دستور pip استفاده کنید:

---

*pip install sympy*

---

برای وارد کردن ماژول SymPy از دستور import استفاده کنید:

---

*import SymPy*

---

## ۵-۱۷) ایجاد متغیرهای نمادین در ماژول SymPy

در SymPy، متغیرهای نمادین یکی از اجزای اصلی و اساسی هستند. این متغیرها به شما اجازه می‌دهند که با متغیرها به صورت انتزاعی و ریاضیاتی کار کنید، بدون اینکه به مقدار عددی خاصی محدود شوند. با استفاده از متغیرهای نمادین می‌توانید عملیات ریاضی مانند مشتق‌گیری، انتگرال‌گیری، حل معادلات و غیره را انجام دهید.

### ۱-۵-۱۷) تعریف متغیرهای نمادین

در SymPy، متغیرهای نمادین با استفاده از تابع `Symbol` یا `symbols()` تعریف می‌شوند. این متغیرها نماینده‌ی یک متغیر ریاضیاتی هستند که می‌توانند در عبارات و محاسبات نمادین استفاده شوند.

### ۲-۵-۱۷) ایجاد متغیر نمادین با `symbols`

#### ۱-۲-۵-۱۷) تعریف متغیر تکی

برای تعریف یک متغیر مانند `x`:

```
from sympy import symbols
```

```
x = symbols('x')
```

### ۲-۲-۵-۱۷) تعریف چندین متغیر

برای تعریف چندین متغیر به صورت همزمان، می‌توانید از آرگومان‌های جداسده با کاما یا رشته‌ای استفاده کنید:

```
from sympy import symbols
```

```
x, y, z = symbols('x y z') # تعریف چند متغیر
```

### ۳-۵-۱۷) ایجاد متغیر با ویژگی‌های خاص در مازول SymPy

متغیرهای نمادین می‌توانند ویژگی‌هایی مانند واقعی، مختلط، مثبت، یا منفی بودن داشته باشند.

### ۱-۳-۵-۱۷) تعریف متغیر مشبّت

```
a = symbols('a', positive=True)
```

### ۲-۳-۵-۱۷) تعریف متغیر واقعی

```
b = symbols('b', real=True)
```

این ویژگی‌ها در محاسبات کمک می‌کنند تا SymPy ساده‌سازی‌های مناسبی انجام دهد.

### ۴-۵-۱۷) ایجاد متغیر با **Symbol**

متغیرها را می‌توان با استفاده از کلاس **Symbol** نیز تعریف کرد. این روش بیشتر برای تعریف متغیرهای با ویژگی خاص استفاده می‌شود:

```
from sympy import Symbol
```

```
x = Symbol('x', positive=True) # متغیر مشبّت
```

### ۵-۵-۱۷) ایجاد آرایه‌ای از متغیرها

برای ایجاد مجموعه‌ای از متغیرها به صورت خودکار (مثلاً  $(X_1, X_2, \dots, X_n)$ )

```
from sympy import Symbol
```

```
variables = symbols('x1:6')
```

## ۶-۱۷) سادهسازی و گسترش عبارات

### ۱-۶-۱۷) سادهسازی عبارات در مژول SymPy

python example - sympy\_module.py

```
1 # SymPy in Python  
2 # Install sympy:  
3 #     pip install sympy  
4  
5 from sympy import symbols, simplify  
6  
7 x = symbols('x')  
8 expr = (x**2 + 2*x + 1)/(x + 1)  
9 print(f"Before = {expr}")  
10  
11 # ساده سازی عبارت جبری  
12 print(f"After = {simplify(expr)}")
```

شکل ۱۷-۱- مثالی از سادهسازی عبارات جبری با استفاده از مژول SymPy

خروجی شکل ۱-۱۷ :

---

*Before = (x\*\*2 + 2\*x + 1)/(x + 1)*

*After = x + 1*

---

## ۱۷-۵-۲) گسترش عبارات در مazzoL SymPy

python example - sympy\_module.py

```
16 # SymPy in Python مژول سیمپای در پایتون
17 # Install sympy: نصب سیمپای
18 #         pip install sympy
19
20 from sympy import symbols, expand
21
22 x = symbols('x')
23 expr = (x + 1)**2
24 print(f"Before = {expr}")
25
26 # گسترش عبارت جبری
27 print(f"After = {expand(expr)}")
```

شکل ۱۷-۲-مثالی از گسترش عبارات جبری با استفاده از مژول SymPy

خروجی شکل ۱۷-۲:

---

*Before = (x + 1)\*\*2*

*After = x\*\*2 + 2\*x + 1*

---

## ۷-۱۷) مبحث حد در مازول SymPy

حد<sup>۱۰۸</sup> یک تابع، مقدار خروجی تابع است زمانی که متغیر ورودی به یک مقدار خاص یا بینهایت نزدیک می‌شود. در SymPy، محاسبه حد یکی از قابلیت‌های اصلی برای انجام محاسبات ریاضی نمادین است. این قابلیت به شما اجازه می‌دهد که حدود توابع را در نقاط مختلف مانند بینهایت یا نقاط خاصی که ممکن است تابع در آن‌ها ناپیوسته باشد، محاسبه کنید.

## ۷-۱۷) محاسبه حد با استفاده از مازول SymPy

برای محاسبه حد در SymPy از تابع limit() استفاده می‌شود:

---

*sympy.limit(expr, var, point, dir)*

---

## ۷-۱۷) محاسبه حد در نقاط مشخص با استفاده از مازول SymPy

محاسبه حد تابع  $f(x)$  زمانی که  $x$  به سمت ۱ میل می‌کند:

---

<sup>108</sup> Limit

### python example - sympy\_module.py

```
31 # SymPy in Python ماثول سیمپای در پایتون
32
33 from sympy import symbols, limit
34 x = symbols('x')
35 expr = (x**2 - 1) / (x - 1)
36 print(f"Equation: {expr}")
37
38 # محاسبه حد تابع
39 result = limit(expr, x, 1)
40 print(f"limit of equation of x as x approaches 1 equals: {result}")
```

شکل ۱۷-۳-۳- مثالی از محاسبه حد زمانی که  $x$  به سمت یک میل می‌کند.

خروجی شکل ۱۷-۳:

---

*Equation: (x\*\*2 - 1)/(x - 1)*

*limit of equation of x as x approaches 1 equals: 2*

---

۱۷-۷-۳) محاسبه حد در بینهایت با استفاده از ماثول SymPy

محاسبه حد تابع  $f(x) = \frac{1}{x}$  زمانی که  $x$  به سمت بینهایت میل می‌کند.

python example - sympy\_module.py

```
44 # SymPy in Python ماثول سیمپای در پایتون
45
46 from sympy import symbols, limit
47 x = symbols('x')
48 expr = 1 / x
49 print(f"Equation: {expr}")
50
51 # محاسبه حد تابع
52 result = limit(expr, x, float('inf'))
53 print(f"limit of equation of x as x approaches infinity equals: {result}")
```

شکل ۱۷-۴-مثالی از محاسبه حد زمانی که  $x$  به سمت بی‌نهایت میل می‌کند.

خروجی شکل ۱۷-۴:

---

*Equation: 1/x*

*limit of equation of x as x approaches infinity equals: 0*

---

۱۷-۷-۴) محاسبه حد از چپ و راست با استفاده از ماثول SymPy

برای محاسبه حد از راست یا چپ:

حد تابع  $f(x) = \frac{1}{x}$  وقتی که  $x$  به سمت  $0^+$  (صفر مثبت) میل کند:

python example - sympy\_module.py

```
57 # SymPy in Python مازول سیمپای در پایتون
58
59 from sympy import symbols, limit
60 x = symbols('x')
61 expr = 1 / x
62 print(f"Equation: {expr}")
63
64 # One-sided limit
65 # محاسبه حد یک طرفه تابع
66 result = limit(expr, x, 0, dir='+')
67 print(f"limit of equation of x as x approaches a 'from the right': {result}")
```

شکل ۱۷-۵- مثالی از محاسبه حد زمانی که  $x$  به سمت صفر مثبت میل می‌کند.

خروجی شکل ۱۷-۵:

---

*Equation: 1/x*

*limit of equation of x as x approaches a 'from the right':  $\infty$*

---

- حد همان تابع وقتی  $x$  به سمت  $0^-$  (صفر منفی) میل کند:

### python example - sympy\_module.py

```
71 # SymPy in Python مازول سیمپای در پایتون
72
73 from sympy import symbols, limit
74 x = symbols('x')
75 expr = 1 / x
76 print(f"Equation: {expr}")
77
78 # One-sided limit
79 # محاسبه حد یک طرفه تابع
80 result = limit(expr, x, 0, dir='-')
81 print(f"limit of equation of x as x approaches a 'from the left': {result}")
```

شکل ۱۷-۶- مثالی از محاسبه حد زمانی که  $x$  به سمت صفر منفی میل می‌کند.

خروجی شکل ۱۷-۶:

---

*Equation: 1/x*

*limit of equation of x as x approaches a 'from the left': -∞*

---

۱۷-۷- (۵) حد توابع مثلثاتی با استفاده از مازول SymPy

حد تابع  $f(x) = \sin(x)/x$  زمانی که  $x$  به سمت صفر میل کند.

### python example - sympy\_module.py

```
85 # SymPy in Python مژول سیمپای در پایتون
86
87 from sympy import symbols, limit, sin
88 x = symbols('x')
89 expr = sin(x)/x
90 print(f"Equation: {expr}")
91
92 # محاسبه حد توابع مثلثاتی
93 result = limit(expr, x, 0)
94 print(f"limit of equation of x as x approaches 0 equals: {result}")
```

شکل ۱۷-۷- مثالی از محاسبه حد توابع مثلثاتی زمانی که  $x$  به سمت صفر میل می‌کند

خروجی شکل ۱۷-۷:

---

*Equation: sin(x)/x*

*limit of equation of x as x approaches 0 equals: 1*

---

۱۷-۶) بررسی حد برای نقاط ناپیوسته با استفاده از مژول SymPy

برای توابعی که در نقاط خاص ناپیوستگی دارند، SymPy می‌تواند بررسی کند که حد وجود دارد یا خیر:

### python example - sympy\_module.py

```
98 # SymPy in Python ماثول سیمپای در پایتون
99
100 from sympy import symbols, limit, sin
101 x = symbols('x')
102 expr = 1/(x-1)
103 print(f"Equation: {expr}")
104
105 # حد در نقاط ناپیوسته
106 left_limit = limit(expr, x, 1, dir='-') # حد از چپ
107 right_limit = limit(expr, x, 1, dir='+') # حد از راست
108
109 print(f"limit of equation of x as x approaches 1 'from the right': {left_limit}")
110 print(f"limit of equation of x as x approaches 1 'from the left': {right_limit}")
```

شکل ۱۷-۸- مثالی از محاسبه حد برای نقاط ناپیوسته

خروجی شکل ۱۷-۸:

---

*limit of equation of x as x approaches 1 'from the right': -∞*

*limit of equation of x as x approaches 1 'from the left': ∞*

---

## ۱۷-۸) مبحث مشتق در ماثول SymPy

در SymPy، مبحث مشتق‌گیری یکی از امکانات اصلی برای انجام محاسبات نمادین است. با استفاده از این قابلیت می‌توان مشتقات توابع ساده و پیچیده، توابع چندمتغیره، و حتی مشتقات بالاتر را به راحتی محاسبه کرد.

مشتق، نرخ تغییرات یک تابع نسبت به یک متغیر است. به طور کلی:

- مشتق اول: سرعت تغییرات مقدار تابع.

- مشتق دوم و بالاتر: تغییرات سرعت یا نرخ تغییرات توابع.

## ۱-۸-۱۷) محاسبه مشتق با استفاده از ماثول SymPy

برای مشتق‌گیری از تابع `diff()` استفاده می‌شود:

---

`sympy.diff(expr, var, n)`

---

## ۲-۸-۱۷) محاسبه مشتق اول با استفاده از ماثول SymPy

مثال: محاسبه مشتق اول  $f(x) = x^2$

python example - sympy\_module.py

```

114 # SymPy in Python در پایتون
115
116 from sympy import symbols, diff
117 x = symbols('x')
118 expr = x**2
119 print(f"Equation: {expr}")
120
121 # محاسبه مشتق اول
122 print(f"Differential of {expr} is: \n\t{diff(expr, x)}")

```

شکل ۱-۹-۱۷- مثالی از محاسبه مشتق اول توسط ماثول SymPy

خروجی شکل ۱-۹-۱۷:

---

*Equation: x\*\*2*

*Differntial of  $x^{**}2$  is:*

$2*x$

---

۱۷-۸-۳) محاسبه مشتق بالاتر با استفاده از مازول SymPy

مثال: محاسبه مشتق دوم  $f(x) = x^3$

python example - sympy\_module.py

```
126 # SymPy in Python مازول سیمپای در پایتون
127
128 from sympy import symbols, diff
129 x = symbols('x')
130 expr = x**3
131 print(f"Equation: {expr}")
132
133 # محاسبه مشتق دوم
134 print(f"Differntial of {expr} is: \n\t{diff(expr, x, 2)}")
```

شکل ۱۰-۱۷- مثالی از محاسبه مشتق دوم توسط مازول SymPy

: ۱۰-۱۷ خروجی شکل

---

*Equation:  $x^{**}3$*

*Differntial of  $x^{**}3$  is:*

$6*x$

---

## ۴-۸-۱۷) محاسبه مشتق توابع چندمتغیره با استفاده از مازول SymPy

می‌تواند مشتقات جزئی توابع چندمتغیره را محاسبه کند.

$$f(x, y) = x^2 + y^2 \text{ برای } \frac{\partial}{\partial y} \text{ و } \frac{\partial}{\partial x}$$

python example - sympy\_module.py

```
138 # SymPy in Python مازول سیمپای در پایتون
139
140 from sympy import symbols, diff
141 x = symbols('x')
142 y = symbols('y')
143 expr = x**2 + y**2
144 print(f"Equation: {expr}")
145
146 partial_x = diff(expr, x) # نسبت به x
147 partial_y = diff(expr, y) # نسبت به y
148
149 محاسبه مشتق توابع چند متغیره #
150 print(f"Differntial of partial_x is: \t{partial_x}")
151 print(f"Differntial of partial_y is: \t{partial_y}")
```

شکل ۱۷-۱۱-۱۱- مثالی از محاسبه مشتق توابع چند متغیره توسط مازول SymPy

: ۱۱-۱۷ خروجی شکل

---

*Equation:  $x^{**2} + y^{**2}$*

*Differntial of partial\_x is:  $2*x$*

*Differntial of partial\_y is:  $2*y$*

---

## ۱۷-۸-۵) محاسبه مشتق توابع مثلثاتی با استفاده از ماژول SymPy

از توابع مثلثاتی نیز پشتیبانی می‌کند.

مثال: مشتق  $f(x) = \sin(x)$

python example - sympy\_module.py

```
155 # SymPy in Python
156
157 from sympy import symbols, diff, sin
158
159 x = symbols('x')
160 expr = sin(x)
161 print(f"Equation: {expr}")
162
163 محاسبه مشتق توابع مثلثاتی #
164 result = diff(expr, x)
165 print(result)
```

شکل ۱۷-۱۲-۱۲- مثالی از محاسبه مشتق توابع مثلثاتی توسط ماژول SymPy

خروجی شکل ۱۷-۱۲:

---

*Equation: sin(x)*

*cos(x)*

---

## ۱۷-۸-۶) محاسبه مشتق توابع مرکب با استفاده از مازول SymPy

به طور خودکار قانون زنجیره‌ای<sup>۱۰۹</sup> را اعمال می‌کند.

مثال: مشتق  $f(x) = \sin(x^2)$

python example - sympy\_module.py

```
169 # SymPy in Python مازول سیمپای در پایتون
170
171 from sympy import symbols, diff, sin
172
173 x = symbols('x')
174 expr = sin(x**2)
175 print(f"Equation: {expr}")
176
177 محاسبه مشتق توابع مرکب #  

178 result = diff(expr, x)
179 print(result)
```

شکل ۱۳-۱۷-۱۳- مثالی از محاسبه مشتق توابع مرکب توسط مازول SymPy

خروجی شکل ۱۳-۱۷:

---

Equation:  $\sin(x^{**2})$

$2*x*cos(x^{**2})$

---

<sup>109</sup> Chain Rule

## ۹-۱۷) مبحث انتگرال در ماژول SymPy

در SymPy، مبحث انتگرال یکی از قابلیت‌های کلیدی برای انجام محاسبات نمادین است. با استفاده از این قابلیت می‌توان انتگرال توابع مختلف را به صورت تحلیلی (تحلیل ریاضی) یا عددی محاسبه کرد. SymPy از ابزارهای پیشرفته‌ای برای انجام انتگرال‌های ساده و پیچیده پشتیبانی می‌کند.

### ۱-۹-۱۷) تعریف انتگرال نامعین

انتگرال نامعین<sup>۱۱۰</sup> عبارت است از تابعی که مشتق آن برابر با تابع اولیه باشد.

$$\int f(x) dx = F(x) + C$$

که C ثابت انتگرال است.

### ۲-۹-۱۷) تعریف انتگرال معین

انتگرال معین<sup>۱۱۱</sup> مقدار مساحت زیر منحنی تابع  $f(x)$  در بازه  $[a, b]$  را محاسبه می‌کند:

$$\int_a^b f(x) dx$$

### ۳-۹-۱۷) محاسبه انتگرال در SymPy

برای محاسبه انتگرال در SymPy از تابع `integrate()` استفاده می‌شود:

---

`sympy.integrate(expr, var)`

---

<sup>۱۱۰</sup> Indefinite Integral

<sup>۱۱۱</sup> Definite Integral

## ۴-۹-۱۷) محاسبه انتگرال نامعین در مژول SymPy

مثال: محاسبه انتگرال نامعین  $\int x^2 dx$

python example - sympy\_module.py

```
183 # SymPy in Python در پایتون  
184 # Integral in SymPy در سیمپای  
185  
186 from sympy import symbols, integrate  
187 x = symbols('x')  
188 expr = x**2  
189 print(f"Equation: {expr}")  
190  
191 محاسبه انتگرال نامعین #  
192 result = integrate(expr, x)  
193 print(f"Result of Integral: {result}")
```

شکل ۱۷-۱۴-۱۷) مثالی از محاسبه انتگرال نامعین با استفاده از مژول SymPy

خروجی شکل ۱۷-۱۷:

---

*Equation:  $x^{**2}$*

---

*Result of Integral:  $x^{**3}/3$*

---

## ۴-۹-۱۷) محاسبه انتگرال معین در مژول SymPy

مثال: محاسبه انتگرال معین  $\int_0^2 x^2 dx$

python example - sympy\_module.py

```
197 # SymPy in Python مازول سیمپای در پایتون
198 # Integral in SymPy انتگرال در سیمپای
199
200 from sympy import symbols, integrate
201 x = symbols('x')
202 expr = x**2
203 print(f"Equation: {expr}")
204
205 محاسبه انتگرال معین #
206 result = integrate(expr, (x, 0, 2))
207 print(f"Result of Integral: {result}")
```

شکل ۱۷-۱۵- مثالی از محاسبه انتگرال معین با استفاده از مازول SymPy

خروجی شکل ۱۵-۱۷:

---

*Equation:  $x^{**2}$*

*Result of Integral:  $8/3$*

---

۱۷-۹-۶) محاسبه انتگرال توابع مثلثاتی در مازول SymPy

SymPy می‌تواند انتگرال توابع مثلثاتی مانند  $\sin(x)$  یا  $\cos(x)$  را محاسبه کند:

مثال: محاسبه انتگرال  $\sin(x)$  در بازه  $[0, \pi]$

### python example - sympy\_module.py

```
211 # SymPy in Python ماثول سیمپای در پایتون
212 # Integral in SymPy انتگرال در سیمپای
213
214 from sympy import symbols, integrate
215 from sympy import sin, pi
216 x = symbols('x')
217 expr = sin(x)
218 print(f"Equation: {expr}")
219
220 محاسبه انتگرال معین توابع مثلثاتی #
221 result = integrate(expr, (x, 0, pi))
222 print(f"Result of Integral: {result}")
```

شکل ۱۶-۱۷- مثالی از محاسبه انتگرال توابع مثلثاتی با استفاده از ماثول SymPy

خروجی شکل ۱۶-۱۷:

---

*Equation: sin(x)*

---

*Result of Integral: 2*

---

۱۷-۹-۷) محاسبه انتگرال توابع چند متغیره در ماثول SymPy

برای انتگرال توابع چندمتغیره، می‌توان متغیرهای مختلف را به ترتیب مشخص کرد.

مثال: محاسبه  $\int_0^1 \int_0^1 xy \, dx \, dy$

python example - sympy\_module.py

```
228 # SymPy in Python مازول سیمپای در پایتون
229 # Integral in SymPy انگرال در سیمپای
230
231 from sympy import symbols, integrate
232 x = symbols('x')
233 y = symbols('y')
234 expr = x * y
235 print(f"Equation: {expr}")
236
237 محاسبه انگرال توابع چند متغیره # ماتریس
238 result = integrate(expr, (x, 0, 1), (y, 0, 1))
239 print(f"Result of Integral: {result}")
```

شکل ۱۷-۱۷- مثالی از محاسبه انگرال توابع چند متغیره با استفاده از مازول *SymPy*

خروجی شکل ۱۷-۱۷:

---

*Equation: x\*y*

*Result of Integral: 1/4*

---

۱۷-۸) محاسبه انگرال عددی در مازول *SymPy*

برای محاسباتی که پاسخ به صورت نمادین ممکن نیست، می‌توان از روش‌های عددی استفاده کرد:

python example - sympy\_module.py

```
244 # SymPy in Python مازول سیمپای در پایتون
245 # Integral in SymPy انتگرال در سیمپای
246
247 from sympy import N
248 from sympy import symbols, integrate, sin
249 x = symbols('x')
250 expr = integrate(sin(x) / x, (x, 1, 10))
251 print(f"Equation: {expr}")
252
253 # محاسبه انتگرال عددی
254 result = N(expr) # تبدیل به عدد تقریبی
255 print(f"Result of Integral: {result}")
```

شکل ۱۷-۱۸-۱۹-مثالی از محاسبه انتگرال عددی با استفاده از مازول SymPy

خروجی شکل ۱۷-۱۸-۱۹:

---

*Equation: -Si(1) + Si(10)*

---

*Result of Integral: 0.712264523851691*

---

۱۷-۱۹-۲۰) محاسبه انتگرال توابع پیچیده در مازول SymPy

از انتگرال گیری از توابع پیچیده نیز پشتیبانی می کند.

مثال: محاسبه  $\int e^{-x^2}$

### python example - sympy\_module.py

```
259 # SymPy in Python مازول سیمپای در پایتون
260 # Integral in SymPy انتگرال در سیمپای
261
262 from sympy import symbols, integrate
263 from sympy import exp
264 x = symbols('x')
265 expr = exp(-x**2)
266 print(f"Equation: {expr}")
267
268 انتگرال گیری از توالع پیچیده #
269 result = integrate(expr, x)
270 print(f"Result of Integral: {result}")
```

شکل ۱۷-۱۹- مثالی از محاسبه انتگرال گیری از توابع پیچیده با استفاده از مازول SymPy

خروج شکل ۱۷-۱۹ :

---

*Equation:  $\exp(-x^{**2})$*

---

*Result of Integral:  $\sqrt{\pi} \cdot \operatorname{erf}(x)/2$*

---

نکته مربوط به شکل ۱۷-۱۹ :

در کتابخانه SymPy، تابع  $\operatorname{erf}(x)$  مخفف Error Function یا تابع خطا است که یک تابع ریاضیاتی مهم در آمار، احتمال، و تحلیل عددی است. این تابع به صورت زیر تعریف می‌شود:

$$\operatorname{erf}(x) = \frac{2}{\sqrt{\pi}} \int_0^x e^{-t^2} dt$$

- این تابع مقدار انتگرال نمایی  $e^{-t^2}$  را از صفر تا  $x$  محاسبه می‌کند.
- تابع خطابه توزیع نرمال مرتبط است و احتمال مقادیر در بازه معین حول میانگین را نشان می‌دهد.

## ۱۰-۱۷) حل معادلات جبری با استفاده از ماژول SymPy

حل معادلات جبری یکی از قابلیت‌های کلیدی SymPy است که به شما امکان می‌دهد معادلات را به صورت نمادین یا عددی حل کنید. SymPy می‌تواند انواع معادلات ساده و پیچیده مانند معادلات خطی، درجه دوم، چندجمله‌ای، و معادلات غیرخطی را حل کند.

## ۱۰-۱۸) توابع حل معادلات در ماژول SymPy

دو تابع اصلی برای حل معادلات جبری دارد:

`solve()`: این تابع برای حل معادلات جبری یا سیستمی از معادلات استفاده می‌شود و ریشه‌های معادله را برمی‌گرداند. ✓

`solveset()`: این تابع جدیدتر و پیشرفته‌تر است و مجموعه جواب‌ها را به صورت دقیق و بهینه‌تر ارائه می‌دهد. ✓

## ۲-۱۰-۱۷) ساختار کلی استفاده از تابع **solve** در مازول SymPy

python example - sympy\_module.py

```
274 # SymPy in Python مازول سیمپای در پایتون
275 # Algebraic equation in SymPy معادلات جبری در سیمپای
276
277 from sympy import symbols, Eq, solve
278 x = symbols('x')
279 expr = Eq(expression1, expression2) # معادله‌ای به فرم expression1 = expression2
280 solution = solve(expr, x)
```

شکل ۲۰-۲۰- ساختار کلی استفاده از تابع *solve* برای حل معادلات جبری

## ۳-۱۰-۱۷) حل معادلات ساده با استفاده از مازول SymPy

۱-۳-۱۰-۱۷) حل یک معادله خطی

مثال: حل معادله  $x+2=0$

python example - sympy\_module.py

```
284 # SymPy in Python مازول سیمپای در پایتون
285 # Algebraic equation in SymPy معادلات جبری در سیمپای
286
287 from sympy import symbols, Eq, solve
288 x = symbols('x')
289 expr = x + 2
290 print(f"Equation: {expr}")
291
292 # حل معادله خطی
293 solution = solve(expr, x) # معادله به صورت  $x + 2 = 0$ 
294 print(f"Result: {solution}")
```

شکل ۲۱-۲۱- مثالی از حل معادله خطی با استفاده از مازول SymPy

خروجی شکل ۱۷-۲۱:

*Equation:  $x + 2$*

*Result: [-2]*

۱۷-۱۰-۳-۲) حل یک معادله درجه دوم

مثال: حل معادله  $x^2 - 4x + 3 = 0$

python example - sympy\_module.py

```
298 # SymPy in Python مازول سیمپای در پایتون
299 # Algebraic equation in SymPy معادلات جبری در سیمپای
300
301 from sympy import symbols, Eq, solve
302 x = symbols('x')
303 expr = x**2 - 4*x + 3
304 print(f"Equation: {expr}")
305
306 حل معادله درجه دوم #
307 solution = solve(expr, x)
308 print(f"Result: {solution}")
```

شکل ۱۷-۲۲-مثالی از حل معادله درجه دوم با استفاده از مازول SymPy

خروجی شکل ۱۷-۲۲:

*Equation:  $x^{**2} - 4*x + 3$*

*Result: [1, 3]*

## ۴-۱۰-۱۷) حل معادلات چندمتغیره با استفاده از ماژول **Sympy**

### ۱-۴-۱۰-۱۷) یک سیستم معادلات

مثال: حل سیستم معادلات

$$x + y = 3$$

$$x - y = 1$$

python example - sympy\_module.py

```
ماژول سیمپای در پایتون
معادلات جبری در سیمپای SymPy
312 # SymPy in Python
313 # Algebraic equation in SymPy
314
315 from sympy import symbols, Eq, solve
316 x, y = symbols('x y')
317
318 eq1 = Eq(x + y, 3)
319 print(f"Equation 1: {eq1}")
320
321 eq2 = Eq(x - y, 1)
322 print(f"Equation 2: {eq2}")
323
324 # حل سیستم معادلات
325 solution = solve([eq1, eq2], (x, y))
326 print(f"Result: {solution}")
```

شکل ۱۷-۲۳- مثالی از حل سیستم معادلات با استفاده از ماژول **Sympy**

خروجی شکل ۱۷-۲۳:

---

*Equation 1: Eq(x + y, 3)*

*Equation 2: Eq(x - y, 1)*

**Result:** {x: 2, y: 1}

---

## ۱۷-۱۰-۵) حل معادلات غیرخطی با استفاده از ماژول Sympy

می‌تواند معادلات غیرخطی و پیچیده را نیز حل کند.

:مثال

$$x^3 - x + 1 = 0$$

python example - sympy\_module.py

```
330 # SymPy in Python  
331 # Algebraic equation in SymPy  
332  
333 from sympy import symbols, Eq, solve  
334 x = symbols('x')  
335 expr = x**3 - x + 1  
336 print(f"Equation: {expr}")  
337  
338 # حل معادلات غیرخطی  
339 solution = solve(expr, x)  
340 print(f"Result: {solution}")
```

شکل ۱۷-۲۴-۲۴-مثالی از حل معادلات غیرخطی با استفاده از ماژول SymPy

خروجی شکل ۱۷-۲۴-۲۴:

---

*Equation: x\*\*3 - x + 1*

*Result: [-1/((-1/2 - sqrt(3)\*I/2)\*(3\*sqrt(69)/2 + 27/2)\*\*(1/3)) - (-1/2 - sqrt(3)\*I/2)\*(3\*sqrt(69)/2 + 27/2)\*\*(1/3)/3, -(-1/2 + sqrt(3)\*I/2)\*(3\*sqrt(69)/2 + 27/2)\*\*(1/3)/3, -(3\*sqrt(69)/2 + 27/2)\*\*(1/3)/3 - 1/(3\*sqrt(69)/2 + 27/2)\*\*(1/3)]*

---

#### ۱۷-۱۰-۶) حل معادلات با solveset

solveset() به جای فهرست جواب‌ها، مجموعه‌ی جواب‌ها را برمی‌گرداند.

مثال: حل معادله  $x^2 - 4 = 0$

python example - sympy\_module.py

```

344 # SymPy in Python
345 # Algebraic equation in SymPy
346
347 from sympy import solveset, S
348 from sympy import symbols, Eq, solve
349 x = symbols('x')
350 expr = x**2 - 4
351 print(f"Equation: {expr}")
352
353 # solveset با روشن
354 solution = solveset(expr, x, domain=S.Reals)
355 print(f"Result: {solution}")

```

شکل ۱۷-۲۵- مثالی از حل معادلات با solveset در مازول SymPy

:۲۵-۱۷ خروجی شکل

---

*Equation:*  $x^{**2} - 4$

---

*Result:*  $\{-2, 2\}$

---

## ۱۱-۱۷) حل معادلات دیفرانسیل با استفاده از ماژول SymPy

حل معادلات دیفرانسیل یکی از قابلیت‌های پیشرفته و کلیدی SymPy است. این ماژول با ارائه ابزارهای قوی برای حل معادلات دیفرانسیل معمولی و حتی معادلات دیفرانسیل جزئی، امکان انجام تحلیل‌های ریاضی را به صورت نمادین و عددی فراهم می‌کند.

## ۱۱-۱۸) تعریف معادله دیفرانسیل در SymPy

معادلات دیفرانسیل معمولی با استفاده از تابع `Eq()` تعریف می‌شوند و سپس با استفاده از `dsolve()` حل می‌شوند. برای تعریف مشتق‌ها، از `Derivative` یا نمادگذاری خلاصه `diff` استفاده می‌شود.

## ۲-۱۱-۱۷) توابع اصلی برای حل معادلات دیفرانسیل

### ۱-۲-۱۱-۱۷) `dsolve()` تابع

این تابع معادلات دیفرانسیل معمولی را حل می‌کند. ساختار کلی:

---

`dsolve(eq, func, hint)`

---

### ۱۷-۱۱-۳) حل معادلات دیفرانسیل ساده

مثال: معادله  $\frac{dy}{dx} = x$

python example - sympy\_module.py

```
359 # SymPy in Python
360 # Differential equation in SymPy
361 from sympy import symbols, Function, Eq, dsolve
362
363 x = symbols('x')
364 y = Function('y')(x)
365 expr = y.diff(x)
366 print(f"Equation: {expr}")
367
368 حل معادله دیفرانسیل ساده
369 eq = Eq(expr, x)
370 solution = dsolve(eq, y)
371 print(f"Result: {solution}")
```

شکل ۱۷-۲۶- مثالی از حل معادله دیفرانسیل ساده با استفاده از ماژول SymPy

خروجی شکل ۱۷-۲۶:

---

*Equation:  $y(x)$*

*Result:  $Eq(y(x), C1 + x^{**2}/2)$*

---

### ۱۷-۱۱-۴) حل معادلات دیفرانسیل مرتبه دوم

مثال: معادله  $\frac{d^2y}{dx^2} + 3 \frac{dy}{dx} + 2y = 0$

### python example - sympy\_module.py

```
375 # SymPy in Python  
376 # Differential equation in SymPy  
377 from sympy import symbols, Function, Eq, dsolve  
378  
379 x = symbols('x')  
380 y = Function('y')(x)  
381 expr = y.diff(x, 2) - 3*y.diff(x) + 2*y  
382 print(f"Equation: {expr}")  
383  
384 حل معادله دیفرانسیلی مرتبه دوم #  
385 eq = Eq(expr, 0)  
386 solution = dsolve(eq, y)  
387 print(f"Result: {solution}")
```

شکل ۱۷-۲۷- مثالی از حل معادله دیفرانسیلی مرتبه دوم با استفاده از ماژول SymPy

خروجی شکل ۱۷-۲۷:

---

*Equation: 2\*y(x) - 3\*Derivative(y(x), x) + Derivative(y(x), (x, 2))*

*Result: Eq(y(x), (C1 + C2\*exp(x))\*exp(x))*

---

۱۷-۱۱-۵) معادلات دیفرانسیل غیرخطی

SymPy می‌تواند برخی از معادلات دیفرانسیل غیرخطی را نیز حل کند.

مثال: معادله  $\frac{dy}{dx} = y^2$

### python example - sympy\_module.py

```
391 # ماژول سیمپای در پایتون
392 # Differential equation in SymPy
393
394 from sympy import symbols, Function, Eq, dsolve
395 x = symbols('x')
396 y = Function('y')(x)
397
398 حل معادلات دیفرانسیلی غیر خطی #
399 eq = Eq(y.diff(x), y**2)
400 solution = dsolve(eq, y)
401 print(f"Result: {solution}")
```

شکل ۱۷-۲۸- مثالی از حل معادلات دیفرانسیلی غیرخطی با استفاده از ماژول *SymPy*

خروجی شکل ۱۷-۲۸:

---

*Result: Eq(y(x), -1/(C1 + x))*

---

## ۱۷-۱۲) مثال‌های کاربردی **SymPy**

در اینجا چندین مثال کاربردی از *SymPy* برای نشان دادن توانایی‌های آن در مسائل ریاضی آورده شده است:

### ۱۷-۱۲-۱) ساده‌سازی عبارات ریاضی

مثال: با استفاده از ماژول *SymPy*, عبارت  $y(x) = \frac{x^2+2x+1}{x+1}$  را ساده کنید.

### python example - sympy\_module.py

```
405 # SymPy in Python  
406 # SymPy Examples  
407  
408 from sympy import symbols, simplify  
409 x = symbols('x')  
410 expr = (x**2 + 2*x + 1) / (x + 1)  
411 result = simplify(expr)  
412 print(result)
```

شکل ۱۷-۲۹- مثال مربوط به سادهسازی عبارت ریاضی

خروجی شکل ۱۷-۲۹:

---

$x + 1$

---

۱۷-۱۲-۲) حل معادلات جبری (حل معادله درجه دوم)

مثال: با استفاده از ماژول SymPy، معادله درجه دوم  $x^2 - 4$  را حل کنید.

### python example - sympy\_module.py

```
417 # SymPy in Python  
418 # SymPy Examples  
419  
420 from sympy import symbols, solve  
421 x = symbols('x')  
422 eq = x**2 - 4  
423 roots = solve(eq, x)  
424 print(roots)
```

شکل ۱۷-۳۰- مثال مربوط به حل معادله درجه دوم

خروجی شکل ۱۷-۳۰:

---

$[-2, 2]$

---

۱۷-۱۲-۳) محاسبه مشتق یک تابع

مثال: با استفاده از ماژول SymPy، مشتق تابع  $y(x) = x^3 + 2x^2 + x$  را محاسبه کنید.

### python example - sympy\_module.py

```
428 # SymPy in Python  
429 # SymPy Examples  
430  
431 from sympy import symbols, diff  
432 x = symbols('x')  
433 expr = x**3 + 2*x**2 + x  
434 derivative = diff(expr, x)  
435 print(derivative)
```

شکل ۱۷-۳۱- مثال مربوط به محاسبه مشتق یک تابع

خروجی شکل ۱۷-۳۱:

---

$$3x^2 + 4x + 1$$

---

۱۷-۱۲-۴) انتگرال‌گیری از یک تابع

مثال: با استفاده از ماژول SymPy، انتگرال تابع  $y(x) = x^2$  را محاسبه کنید.

### python example - sympy\_module.py

```
439 # SymPy in Python  
440 # SymPy Examples  
441  
442 from sympy import symbols, integrate  
443 x = symbols('x')  
444 expr = x**2  
445 integral = integrate(expr, x)  
446 print(integral)
```

شکل ۱۷-۳۲- مثال مربوط به محاسبه انتگرال یک تابع

خروجی شکل ۱۷-۳۲:

---

$x^{3/2}$

---

۱۷-۱۲-۵) سری تیلور

مثال: با استفاده از ماژول SymPy، سری تیلور را پیاده‌سازی کنید.

### python example - sympy\_module.py

```
450 # SymPy in Python  
451 # SymPy Examples  
452  
453 #پیاده سازی سری تیلور  
454 from sympy import symbols, sin, series  
455 x = symbols('x')  
456 expr = sin(x)  
457 taylor = series(expr, x, 0, 6)  
458 print(taylor)
```

شکل ۱۷-۳۳-مثال مربوط به پیاده سازی سری تیلور در ماژول *Sympy*

خروجی شکل ۱۷-۳۳:

---

$$x - x^{**3}/6 + x^{**5}/120 + O(x^{**6})$$

---

۱۷-۱۲-۶) کاربرد در هندسه (محاسبه مساحت دایره)

مثال: با استفاده از ماژول *Sympy*, مساحت دایره را به صورت تحلیلی، پیاده سازی کنید.

## python example - sympy\_module.py

```
462 # SymPy in Python ماثول سیمپای در پایتون  
463 # SymPy Examples مثال های مربوط به ماثول سیمپای  
464  
465 # پیاده سازی مساحت دایره به صورت تحلیلی  
466 from sympy import symbols, pi  
467 r = symbols('r')  
468 area = pi * r**2  
469 print(area)  
470
```

شکل ۱۷-۳۴- مثال مربوط به پیاده سازی مساحت دایره به صورت تحلیلی در ماثول *SymPy*

خروجی شکل ۱۷-۳۴:

---

*pi\*r\*\*2*

---

# فصل هجدهم

آشنایی با ماثول pandas در پایتون

## ۱۸) آشنایی با ماژول **pandas** در پایتون

یک کتابخانه متن باز و قدرتمند در زبان برنامه‌نویسی پایتون است که برای تحلیل داده‌ها و دستکاری آن‌ها طراحی شده است. این کتابخانه به طور گسترده در علم داده، یادگیری ماشین، و تحلیل آماری استفاده می‌شود و به دلیل سهولت استفاده و قابلیت‌های متعدد، یکی از محبوب‌ترین ابزارها در جامعه پایتون به شمار می‌رود.

### ۱-۱۸) **Pandas** ویژگی‌های اصلی

#### ۱. ساختار داده‌های کارآمد

یک جدول دو بعدی مشابه صفحات گسترده (مانند Excel) با ردیف‌ها و

ستون‌هایی که می‌توانند انواع داده‌های مختلفی را در خود نگه دارند.

یک آرایه یک بعدی که می‌تواند به عنوان یک ستون یا یک ردیف منفرد از یک

DataFrame در نظر گرفته شود.

#### ۲. کاربرد آسان با داده‌ها

بارگذاری داده‌ها از منابع مختلف (مانند فایل‌های JSON، SQL، Excel، CSV و ...).

ذخیره داده‌ها به فرمات‌های مختلف.

مدیریت آسان داده‌های گمشده<sup>۱۱۲</sup>.

#### ۳. ابزارهای تحلیل داده

<sup>۱۱۲</sup> Missing data

- گروه‌بندی<sup>۱۱۳</sup> و تجزیه و تحلیل داده‌ها.
  - عملیات روی داده‌ها مانند مرتب‌سازی، فیلتر کردن، و تبدیل داده‌ها.
  - ادغام و ترکیب داده‌ها از منابع مختلف.
  - پشتیبانی از عملیات پیچیده مانند گروه‌بندی، فیلتر کردن، مرتب‌سازی، و جمع‌آوری داده‌ها.
  - ارائه ابزارهایی برای انجام تحلیل‌های زمانی<sup>۱۱۴</sup>.
  - قابلیت تولید گزارش‌های خلاصه از داده‌ها.
۴. انعطاف‌پذیری بالا
- کار با مجموعه‌های داده کوچک و بزرگ.
  - عملکرد عالی در تحلیل داده‌های زمانی.
  - پشتیبانی از عملیات برداری برای بهبود کارایی.
  - مکان مدیریت داده‌های گمشده یا نامعتبر.
  - تبدیل داده‌ها بین انواع مختلف ساختارها.
  - ادغام و ترکیب مجموعه داده‌ها از منابع گوناگون.
۵. ادغام با سایر ابزارها
- ادغام با کتابخانه‌هایی مانند Scikit-learn ، Matplotlib ، NumPy و

---

<sup>113</sup> Grouping

<sup>114</sup> Time-Series Analysis

- قابلیت تبدیل داده‌ها به ساختارهای دیگر برای استفاده در ابزارهای تخصصی.

#### ۶. کارایی بالا

- این کتابخانه از هسته نامپای بهره می‌برد، که باعث افزایش کارایی در پردازش داده‌ها می‌شود.

- امکان مدیریت داده‌های حجمی بهینه شده برای حافظه.

#### ۷. رابط کاربری ساده

- به‌گونه‌ای طراحی شده که حتی کاربران تازه‌کار نیز بتوانند به راحتی از آن استفاده

کنند، در حالی که ابزارهای پیشرفته‌ای را برای کاربران حرفه‌ای فراهم می‌کند.

### ۲-۱۸) کاربردهای Pandas

۱. تمیز کردن داده‌ها<sup>۱۱۵</sup>: حذف داده‌های گمشده، تصحیح مقادیر نادرست و تبدیل داده‌ها به قالب مناسب.

۲. تحلیل داده‌ها<sup>۱۱۶</sup>: انجام تحلیل‌های آماری و جمع‌آوری اطلاعات مفید از داده‌ها.

۳. کاوش داده‌ها<sup>۱۱۷</sup>: مشاهده خلاصه‌ای از داده‌ها، توزیع مقادیر، و ویژگی‌های کلیدی.

۴. مدیریت داده‌های زمانی: تحلیل و پیش‌بینی داده‌هایی که به زمان وابسته هستند.

۵. پیش‌پردازش داده: حذف داده‌های گمشده، تغییر فرمت ستون‌ها، و ترکیب چند منبع داده برای

آماده‌سازی داده‌ها جهت استفاده در مدل‌های یادگیری ماشین.

<sup>115</sup> Data Cleaning

<sup>116</sup> Data Analysis

<sup>117</sup> Data Exploration

۶. تبدیل داده‌ها: تبدیل داده‌ها به فرمتهای قابل استفاده برای مدل‌های یادگیری ماشین.

### ۳-۱۸) مزایای استفاده از Pandas

- سادگی: Pandas یک API ساده و کاربردی ارائه می‌دهد که استفاده از آن برای مبتدیان آسان است.
- کارایی بالا: این کتابخانه از هسته NumPy برای محاسبات سریع‌تر استفاده می‌کند.
- مستندات جامع و پشتیبانی جامعه: Pandas دارای مستندات کامل است و جامعه بزرگی از کاربران و توسعه‌دهنده‌گان فعال دارد.
- پشتیبانی از داده‌های حجمی: Pandas می‌تواند داده‌های حجمی را که در حافظه قرار می‌گیرند، با کارایی بالا مدیریت کند.

### ۱-۳-۱۸) مقایسه Pandas با سایر مازول‌ها

جدول ۱-۱۸ - جدول مربوط به مقایسه *pandas* با سایر مازول‌ها

| Excel                           | NumPy                   | Pandas                    | ویژگی       |
|---------------------------------|-------------------------|---------------------------|-------------|
| صفحات گسترده                    | آرایه‌های چندبعدی       | ساختارهای داده‌ای پیشرفته | نوع داده‌ها |
| مدیریت داده‌های ساده            | محاسبات عددی            | تحلیل و مدیریت داده‌ها    | کاربرد اصلی |
| بهینه برای داده‌های ساختاریافته | سریع برای داده‌های عددی | محدود به داده‌های کوچک    | عملکرد      |

## ۴-۱۸) نصب و وارد کردن مژول Pandas به پروژه

برای نصب مژول Pandas از دستور زیر استفاده کنید:

---

*pip install pandas*

---

برای وارد کردن مژول pandas به پروژه از دستور زیر استفاده می‌شود:

---

*import pandas*

---

## ۵-۱۸) توابع وابسته‌ی کاربردی در مژول Pandas

در اینجا چند تابع وابسته‌ی کاربردی از Pandas آورده شده است که کاربردهای آن در تحلیل داده را نشان

می‌دهند:

### ۱-۵-۱۸) بارگذاری داده‌ها از فایل CSV و نمایش داده‌ها

مثال:

python example - pandas\_module.py

```
1 # Pandas in Python
2 # Install pandas: نصب پandas
3 #     pip install pandas
4
5 import pandas as pd
6 """
7 pandas.read_csv(filepath, *, sep=<no_default>,
8                 header='infer', names=<no_default>, index_col=None,
9                 usecols=None, dtype=None)
10 """
11 filepath = "C:/Users/Reza Seyyednezhad/Desktop/python/python example/pandas/scores.csv"
12 read_data = pd.read_csv(filepath)
13 print(read_data.head())
```

شکل ۱۸-۱- بارگذاری داده‌ها از فایل CSV

### ۲-۵-۱۸) تمیز کردن داده‌ها (شناسایی و حذف داده‌های گمشده)

برای شناسایی و حذف مقادیر گمشده می‌توانید از طریق کد زیر، این کار را انجام دهید.

---

**حذف ردیف‌هایی که مقادیر گمشده دارند #**

---

### ۳-۵-۱۸) مرتب‌سازی داده‌ها (مرتب‌سازی داده‌ها بر اساس یک ستون)

برای داده‌ها را براساس یک ستون خاص مرتب سازی کنید، لازم است که نام آن ستون را در کد زیر در قسمت column\_name قرار دهید.

---

**df.sort\_values('column\_name', ascending=False, inplace=True)**

---

#### ۱۸-۵-۴) گروهبندی داده‌ها (محاسبه مجموع داده‌ها در گروه‌های مختلف،)

برای گروهبندی داده‌ها طبق کد زیر عمل کنید.

```
grouped = df.groupby('category_column').sum()
```

#### ۸-۵-۴) چرا باید روی داده‌ها گروهبندی انجام دهید؟

گروهبندی در تحلیل داده‌ها یکی از مراحل اساسی است، زیرا به ما اجازه می‌دهد داده‌ها را به دسته‌های مختلف تقسیم کرده و برای هر دسته تحلیل‌های جداگانه‌ای انجام دهیم. دلایل اصلی برای انجام گروهبندی عبارتند از:

۱. تحلیل عمیق‌تر داده‌ها: در داده‌های بزرگ، مشاهده مستقیم کل داده‌ها عموماً دشوار است. گروهبندی به شما این امکان را می‌دهد که به جای تحلیل کلی، بر اساس ویژگی‌های خاصی (مانند دسته‌بندی‌ها، مناطق جغرافیایی، یا زمان) داده‌ها را بررسی کنید. مثال:

- بررسی میانگین فروش بر اساس هر محصول یا دسته‌بندی.
- مقایسه عملکرد کارکنان در بخش‌های مختلف.

۲. محاسبات تجمیعی: گروهبندی به شما امکان می‌دهد تا عملیات‌هایی مانند مجموع، میانگین، ماکسیمم، مینیمم و ... را روی گروه‌های مشخص اعمال کنید. مثال:

- مجموع فروش برای هر ماه.

- میانگین نمرات دانش‌آموزان بر اساس کلاس.

۳. کاهش ابعاد داده: اگر داده‌ها بسیار حجمی هستند، گروهبندی می‌تواند تعداد رکوردها را کاهش دهد و داده‌های ساده‌تر و مفیدتری ارائه دهد. مثال: تبدیل رکوردهای فروش روزانه به مجموع فروش ماهانه.

۴. شناسایی الگوها: گروهبندی به شناسایی الگوهای پنهان در داده‌ها کمک می‌کند. این الگوها می‌توانند برای تصمیم‌گیری‌های استراتژیک مفید باشند. مثال:

- بررسی رفتار خرید مشتریان بر اساس سن یا منطقه.

- شناسایی گروه‌های سنی با بیشترین استفاده از یک محصول خاص.

۵. مقایسه عملکرد گروه‌ها: با گروهبندی می‌توانید عملکرد بخش‌ها یا دسته‌های مختلف را مقایسه کنید. مثال:

- مقایسه میانگین نمرات دانشآموزان در مدارس مختلف.

- بررسی سوددهی محصولات در دسته‌های مختلف.

۶. ایجاد گزارش‌های مدیریتی: در گزارش‌دهی، معمولاً داده‌ها باید بر اساس دسته‌بندی‌های مشخصی (مانند بخش‌ها، محصولات یا مناطق) خلاصه شوند. گروهبندی این نیاز را برطرف می‌کند. مثال: تهیه گزارش فروش محصولات به تفکیک دسته‌بندی برای ارائه به مدیران.

۷. آمده‌سازی داده برای مدل‌سازی: در یادگیری ماشین و مدل‌سازی، داده‌ها اغلب نیاز به پیش‌پردازش دارند. گروهبندی می‌تواند برای ایجاد ویژگی‌های جدید یا خلاصه کردن داده‌ها استفاده شود. مثال: محاسبه میانگین دما برای هر ماه به عنوان یک ویژگی در پیش‌بینی آب و هوای.

۸. تحلیل زمان‌بندی و روندها: گروهبندی بر اساس زمان (روز، ماه، سال) برای شناسایی روندها در داده‌ها استفاده می‌شود. مثال:

- بررسی تغییرات فصلی در فروش.

- شناسایی پیک مصرف انرژی در روزهای مختلف هفته.

۹. تجزیه و تحلیل سفارشی: گاهی نیاز است داده‌ها به صورت دستی بر اساس معیارهای خاصی دسته‌بندی شوند تا نتایج خاصی استخراج شود. مثال: تحلیل مشتریان بر اساس رفتار خرید (مثلاً مشتریان وفادار در مقابل مشتریان معمولی).

۱۸-۵) محاسبات آماری (محاسبه میانگین، کمینه و بیشینه داده‌ها)

```
mean_value = df['numeric_column'].mean() # برای میانگین گیری از داده‌ها  
min_value = df['numeric_column'].min() # کمینه داده‌ها  
max_value = df['numeric_column'].max() # بیشینه داده‌ها
```

۱۸-۶) انتخاب داده‌ها (فیلتر کردن داده‌ها بر اساس یک شرط)

```
filtered_data = df[df['column_name'] > 50]
```

۱۸-۷) ایجاد ستون جدید (ایجاد یک ستون جدید بر اساس محاسبات ستون‌های موجود)

```
df['new_column'] = df['column1'] + df['column2']
```

۱۸-۸) ترکیب و ادغام داده‌ها

```
merged_df = pd.merge(df1, df2, on='common_column')
```

۹-۵-۱۸) تغییر فرمت داده‌ها (تبديل تاریخ‌ها به فرمت زمانی)

---

```
df['date_column'] = pd.to_datetime(df['date_column'])
```

---

۱۰-۵-۱۸) ذخیره داده‌ها (ذخیره داده‌های ویرایش شده به یک فایل جدید)

---

```
df.to_csv('cleaned_data.csv', index=False)
```

---

۱۱-۵-۱۸) محاسبه درصدها (محاسبه نسبت درصدی یک ستون)

---

```
df['percentage'] = (df['column_name'] / df['column_name'].sum()) * 100
```

---

۱۲-۵-۱۸) حذف ستون‌ها

---

```
df.drop('column_name', axis=1, inplace=True)
```

---

۱۳-۵-۱۸) شناسایی و حذف داده‌های تکراری

---

```
df.drop_duplicates(inplace=True)
```

---

این مثال‌ها فقط بخش کوچکی از قابلیت‌های Pandas را نشان می‌دهند. این کتابخانه یکی از قوی‌ترین ابزارها برای تحلیل داده‌ها است که امکانات بسیار گسترده‌ای برای کار با داده‌ها، از تمیز کردن تا تحلیل و مصورسازی، فراهم می‌کند.

## ۶-۱۸) پروژه مربوط به مازول pandas

عنوان پروژه: با استفاده از مازول pandas، میانگین نمرات ۴۰ دانشجو محاسبه کنید و در نهایت در کنار نام و نام خانوادگی دانشجو، میانگین نمرات دروس آن دانشجو را در خروجی چاپ کنید.  
نکته: داده‌های مربوط به دانشجوهای مختلف در کنار کدهای این فصل قرار دارد.

python example - pandas\_module.py

```
17 # Pandas in Python پandas در پایتون
18 import pandas as pd
19
20 # Read Files
21 filepath = "C:/Users/Reza Seyyednezhad/Desktop/python/python example/pandas/scores.csv"
22 read_file = pd.read_csv(filepath, usecols=['id', 'first_name', 'last_name', 'gender',
23                                         'physics_score', 'chemistry_score', 'biology_score',
24                                         'english_score', 'geography_score'])[:40]
25 # print(read_file)
26
27 average_list = []
28
29 for i in range(len(read_file.values)):
30     fullName = read_file.loc[i]['first_name'] + " " + read_file.loc[i]['last_name']
31     avrage_Score = read_file.loc[i][4:].mean()
32     res = {
33         "id": int(read_file.loc[i]['id']),
34         "Full Name": fullName,
35         "Score": float(avrage_Score)
36     }
37     average_list.append(res)
38
39 newDf = pd.DataFrame(average_list)
40 newDf.set_index('id', inplace=True)
41 newDf.to_csv('./newFile.csv')
42 print(newDf)
```

شكل ۶-۲-۲- پروژه مربوط به محاسبه میانگین ۴۰ دانشجو با استفاده از pandas

خروجی شکل ۶-۲:

با گرفتن خروجی، نام و نام خانوادگی دانشجوها به همراه میانگین نمرات دروسشنان چاپ می‌شود.

# فصل نوزدهم

آشنایی با مازول **matplotlib** در پایتون

## ۱۹) آشنایی با ماثول Matplotlib در پایتون

Matplotlib یکی از کتابخانه‌های قدرتمند و محبوب در پایتون است که برای مصورسازی داده‌ها و ایجاد نمودارهای متتنوع استفاده می‌شود. این کتابخانه امکان ایجاد انواع نمودارها از جمله نمودارهای خطی، میله‌ای، دایره‌ای، پراکندگی، هیستوگرام، و بسیاری دیگر را فراهم می‌کند.

### ۱-۱۹) ویژگی‌های Matplotlib

۱. انعطاف‌پذیری بالا: امکان تنظیم دقیق هر بخش از نمودار، از جمله عناوین، برچسب‌های محورها، رنگ‌ها، فونت‌ها، اندازه‌ها، و غیره.

۲. سازگاری با سایر کتابخانه‌ها: Matplotlib به خوبی با کتابخانه‌های محبوب دیگر مانند NumPy و Pandas سازگار است و می‌تواند داده‌ها را به‌طور مستقیم از این کتابخانه‌ها دریافت کند.

۳. پشتیبانی از سبک‌های مختلف نمودار: این کتابخانه به کاربران اجازه می‌دهد نمودارهای مختلفی ایجاد کنند، مانند نمودارهای سه‌بعدی، قطبی، و اینیمیشن‌های پویا.

۴. خروجی گرفتن در فرمتهای مختلف: نمودارها می‌توانند در قالب‌های مختلف از جمله PDF، PNG، SVG، و غیره ذخیره شوند.

۵. پشتیبانی از رابطهای مختلف: Matplotlib می‌تواند از API سطح پایین برای کاربران حرفه‌ای و از رابط ساده‌تر (مانند pyplot) برای کاربران تازه‌کار استفاده کند.

۶. جامع بودن: ابزارهای گسترده‌ای برای ایجاد نمودارهای ساده تا پیچیده ارائه می‌دهد.

۷. منبع‌باز بودن: رایگان است و به‌طور مداوم توسط جامعه کاربران توسعه داده می‌شود.

۸. انعطاف‌پذیری: کاربران می‌توانند نمودارها را به هر شکلی که نیاز دارند، شخصی‌سازی کنند.

۹. ادغام آسان: می‌توان به راحتی از آن در پروژه‌های مختلف علمی، مهندسی، و داده‌محور استفاده کرد.

## ۲-۱۹) زیرماژول‌های Matplotlib

۱. Pyplot: رابطی ساده و مشابه MATLAB برای ایجاد و مدیریت نمودارها که اکثر کاربران از آن استفاده می‌کنند.

۲. mpl\_toolkits.mplot3d: برای ایجاد نمودارهای سه‌بعدی و تحلیل داده‌های سه‌بعدی.

۳. Matplotlib.animation: برای ایجاد انیمیشن‌ها و نمایش‌های پویا از داده‌ها.

## ۳-۱۹) کاربردهای Matplotlib

۱. تحلیل داده‌های علمی: مصورسازی نتایج تجربی، تحلیل روندها، و نمایش داده‌های علمی.

۲. مهندسی: استفاده در شبیه‌سازی‌ها، تحلیل داده‌های تجربی، و ارائه نتایج محاسبات.

۳. یادگیری ماشین و داده‌کاوی: نمایش داده‌ها و مدل‌ها، ارزیابی عملکرد مدل‌ها، و تحلیل ویژگی‌ها.

۴. اقتصاد و مالی: مصورسازی داده‌های بازار، تحلیل روندهای اقتصادی، و نمایش نوسانات.

۵. آموزش و ارائه: استفاده برای تهیه نمودارهای دقیق و حرفه‌ای برای آموزش یا ارائه در کنفرانس‌ها و جلسات.

## ۴-۱۹) ترسیم نمودارها با استفاده از مازول **Matplotlib** در پایتون

رسم نمودار در مازول **Matplotlib** یکی از قابلیت‌های اصلی آن است که به کاربران اجازه می‌دهد داده‌ها را به شیوه‌ای بصری نمایش دهند. با استفاده از زیرماژول **pyplot**، می‌توان به سادگی نمودارهای مختلفی را رسم کرد. در ادامه، مراحل و روش‌های رسم نمودار با ذکر مثال توضیح داده شده‌اند.

### ۱-۴-۱۹) ترسیم نمودار خطی

برای رسم نمودار خطی<sup>۱۱۸</sup> که معمولاً برای نمایش تغییرات یک متغیر استفاده می‌شود:

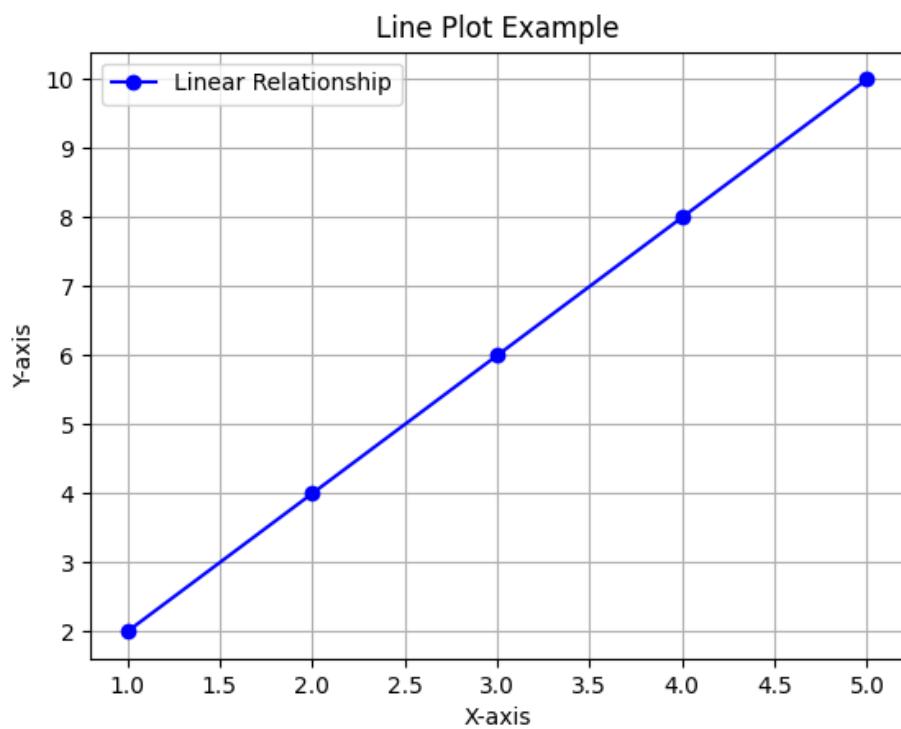
python example - matplotlib\_module.py

```
1 # Matplotlib in Python
2 # Install process: فرایند نصب
3 #      pip install matplotlib
4
5 import matplotlib.pyplot as plt
6
7 x = [1, 2, 3, 4, 5]
8 y = [2, 4, 6, 8, 10]
9
10 plt.plot(x, y, label='Linear Relationship', color='blue', marker='o')
11 plt.title('Line Plot Example') # عنوان نمودار
12 plt.xlabel('X-axis') # برجسب محور X
13 plt.ylabel('Y-axis') # برجسب محور Y
14 plt.legend() # اضافه کردن راهنمای
15 plt.grid(True) # نمایش خطوط شبکه
16 plt.show()
```

شکل ۱-۱۹-۱- مثال مربوط به ترسیم نمودار خطی با استفاده از **matplotlib**

خروجی شکل ۱-۱۹:

<sup>118</sup> Line Plot



شکل ۱۹-۲- خروجی مربوط به نمودار خطی (شکل ۱-۱۹)

۱۱۹) ترسیم نمودار میله‌ای

برای نمایش مقایسه مقادیر دسته‌بندی شده:

---

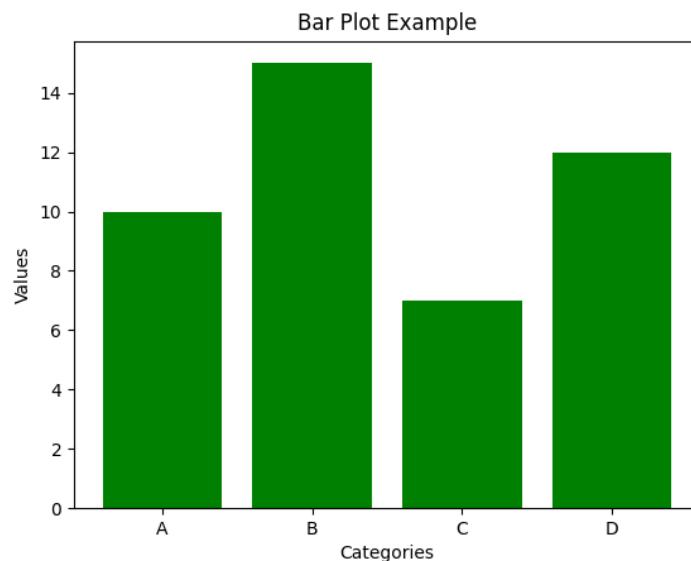
<sup>119</sup> Bar Plot

python example - matplotlib\_module.py

```
20 # Matplotlib in Python  
21  
22 import matplotlib.pyplot as plt  
23  
24 categories = ['A', 'B', 'C', 'D']  
25 values = [10, 15, 7, 12]  
26  
27 plt.bar(categories, values, color='green')  
28 plt.title('Bar Plot Example')  
29 plt.xlabel('Categories')  
30 plt.ylabel('Values')  
31 plt.show()
```

شکل ۱۹-۳- مثال مربوط به ترسیم نمودار میله‌ای با استفاده از *matplotlib*

خروجی شکل ۱۹-۲:



شکل ۱۹-۴- خروجی مربوط به نمودار میله‌ای (شکل ۱۹-۳)

## ۱۹-۴-۳) ترسیم نمودار دایره‌ای<sup>۱۲۰</sup>

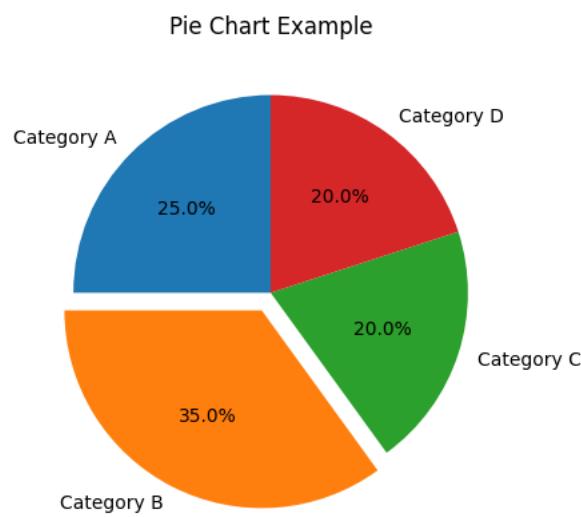
برای نمایش نسبت‌ها و درصدها:

python example - matplotlib\_module.py

```
35 # Matplotlib in Python
36 import matplotlib.pyplot as plt
37
38 labels = ['Category A', 'Category B', 'Category C', 'Category D']
39 sizes = [25, 35, 20, 20]
40 explode = (0, 0.1, 0, 0) # برجسته کردن بخش دوم
41
42 plt.pie(sizes, labels=labels, autopct='%1.1f%%', explode=explode, startangle=90)
43 plt.title('Pie Chart Example')
44 plt.show()
45
```

شکل ۱۹-۵- مثال مربوط به ترسیم نمودار دایره‌ای با استفاده از *matplotlib*

خروجی شکل ۱۹-۶:



شکل ۱۹-۶- خروجی مربوط به نمودار دایره‌ای (شکل ۱۹-۵)

<sup>120</sup> Pie Chart

## ۱۹-۴-۴) ترسیم نمودار پراکندگی<sup>۱۲۱</sup>

برای نمایش رابطه بین دو متغیر:

python example - matplotlib\_module.py

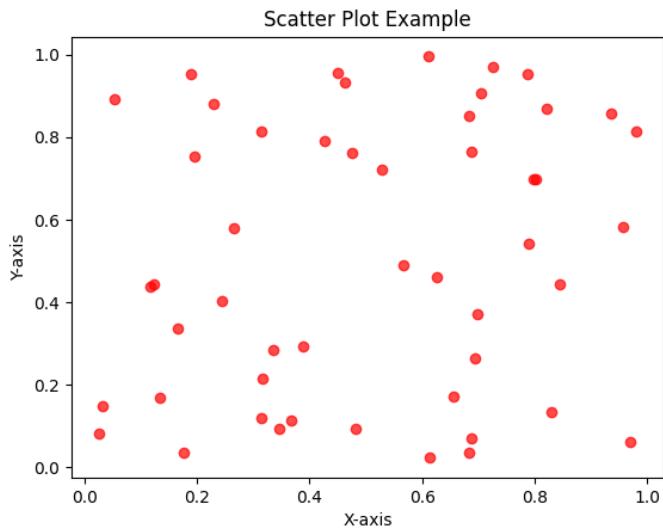
```
48 # Matplotlib in Python
49 import numpy as np
50 import matplotlib.pyplot as plt
51
52 x = np.random.rand(50)
53 y = np.random.rand(50)
54
55 plt.scatter(x, y, color='red', alpha=0.7)
56 plt.title('Scatter Plot Example')
57 plt.xlabel('X-axis')
58 plt.ylabel('Y-axis')
59 plt.show()
```

شکل ۱۹-۷- مثال مربوط به ترسیم نمودار پراکندگی با استفاده از *matplotlib*

خروجی شکل ۱۹-۷:

---

<sup>۱۲۱</sup> Scatter Plot



شکل ۱۹-۸- خروجی مربوط به نمودار پراکندگی (شکل ۷-۱۹)

۱۲۲) ترسیم هیستوگرام

برای نمایش توزیع داده‌ها:

python example - matplotlib\_module.py

```

63 # Matplotlib in Python
64
65 import matplotlib.pyplot as plt
66 data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5]
67
68 plt.hist(data, bins=5, color='purple', edgecolor='black')
69 plt.title('Histogram Example')
70 plt.xlabel('Value')
71 plt.ylabel('Frequency')
72 plt.show()

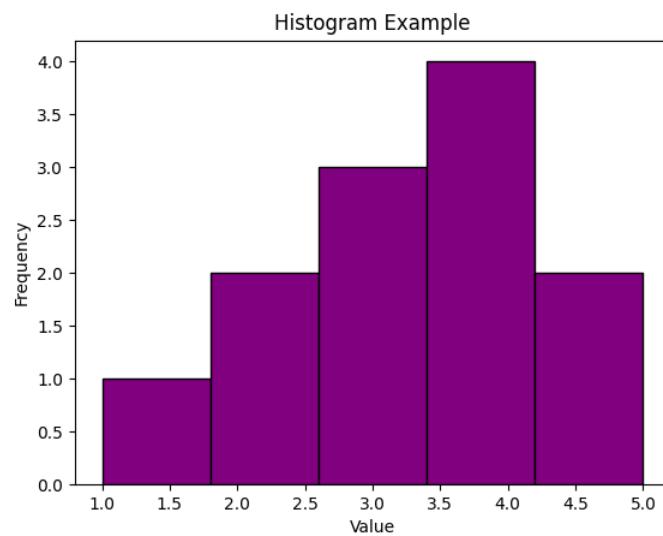
```

شکل ۱۹-۹- مثال مربوط به ترسیم نمودار هیستوگرام با استفاده از *matplotlib*

---

<sup>122</sup> Histogram

خروجی شکل ۹-۱۹:



شکل ۹-۱۰ - خروجی مربوط به نمودار هیستوگرام (شکل ۹-۱۹)

۶-۴-۱۹) ترسیم چند نمودار در یک صفحه

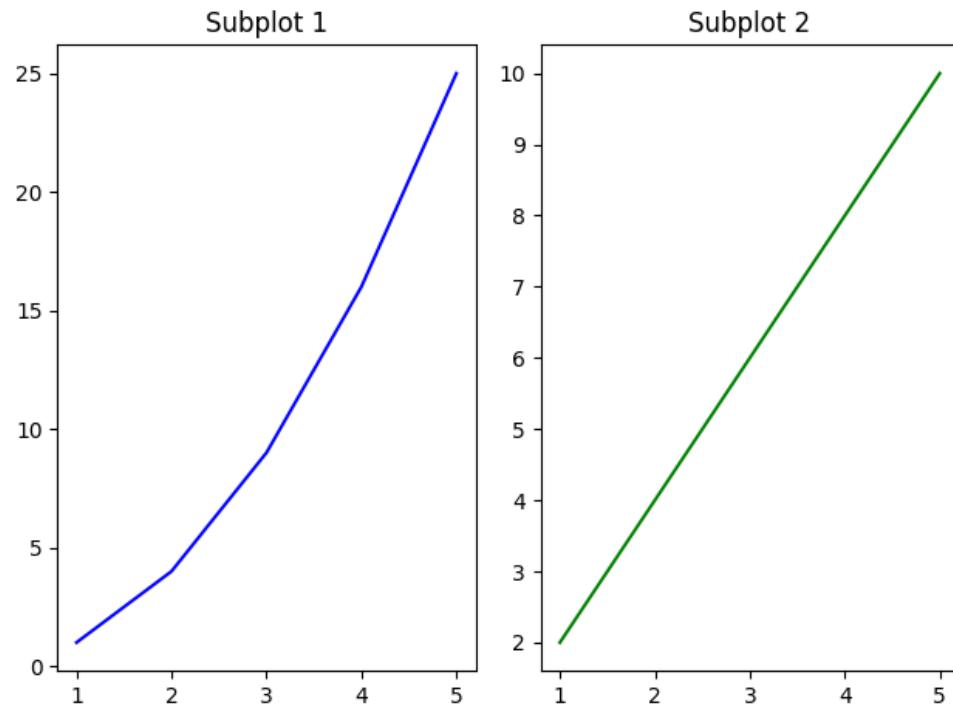
برای نمایش چند نمودار به صورت جداگانه در یک صفحه:

### python example - matplotlib\_module.py

```
76 # Matplotlib in Python
77 import matplotlib.pyplot as plt
78
79 x = [1, 2, 3, 4, 5]
80 y1 = [1, 4, 9, 16, 25]
81 y2 = [2, 4, 6, 8, 10]
82
83 plt.subplot(1, 2, 1) # يک ردیف، دو ستون، نمودار اول
84 plt.plot(x, y1, color='blue')
85 plt.title('Subplot 1')
86
87 plt.subplot(1, 2, 2) # يک ردیف، دو ستون، نمودار دوم
88 plt.plot(x, y2, color='green')
89 plt.title('Subplot 2')
90
91 plt.tight_layout() # تنظیم فاصله‌ها
92 plt.show()
```

شکل ۱۱-۱۹- مثالی از ترسیم چند نمودار در یک صفحه در *matplotlib*

خروجی شکل ۱۱-۱۹:



شکل ۱۹-۱۲- خروجی مربوط به ترسیم چند نمودار (شکل ۱۹-۱۱-۱۲)

۱۹-۴-۷) ترسیم نمودار سه بعدی<sup>۱۲۳</sup>

برای نمایش داده های سه بعدی:

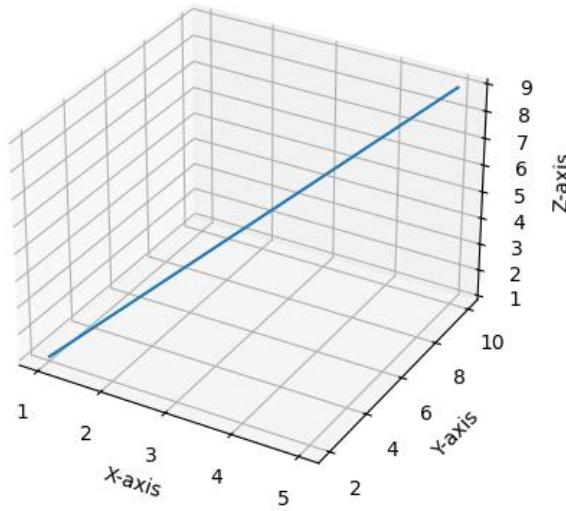
python example - matplotlib\_module.py

```
96 # Matplotlib in Python  
97  
98 import matplotlib.pyplot as plt  
99 from mpl_toolkits.mplot3d import Axes3D  
100  
101 fig = plt.figure()  
102 ax = fig.add_subplot(111, projection='3d')  
103  
104 x = [1, 2, 3, 4, 5]  
105 y = [2, 4, 6, 8, 10]  
106 z = [1, 3, 5, 7, 9]  
107  
108 ax.plot(x, y, z, label='3D Line')  
109 ax.set_title('3D Plot Example')  
110 ax.set_xlabel('X-axis')  
111 ax.set_ylabel('Y-axis')  
112 ax.set_zlabel('Z-axis')  
113 plt.show()
```

شکل ۱۹-۱۳- مثال مربوط به ترسیم نمودار سه بعدی در *matplotlib*

خروجی شکل ۱۹-۱۳:

### 3D Plot Example



شکل ۱۹-۱۴-۱۹- خروجی مربوط به ترسیم نمودار سه بعدی (شکل ۱۹-۱۹)

### ۸-۴-۱۹) ترسیم نمودارهای log-log در مازول matplotlib

نمودارهای log-log در مازول Matplotlib به شما امکان می دهند که داده هایی که به صورت نمایی یا لگاریتمی رشد می کنند را بهتر تحلیل و نمایش دهید. این نوع نمودارها زمانی مفید هستند که داده ها دارای مقادیر بسیار بزرگ یا کوچک باشند یا رابطه هی توان بین متغیرها وجود داشته باشد.

### ۱-۸-۴-۱۹) روش رسم نمودار log-log

در Matplotlib می توانید از دو روش برای رسم نمودارهای log-log استفاده کنید:

- استفاده از تابع `loglog`: تابع `loglog` هر دو محور `X` و `Y` را به مقیاس لگاریتمی تنظیم می کند.
- تنظیم دستی مقیاس محورها: می توانید از توابع `set_yscale` و `set_xscale` برای تنظیم مقیاس محورهای `X` و `Y` به صورت لگاریتمی استفاده کنید.

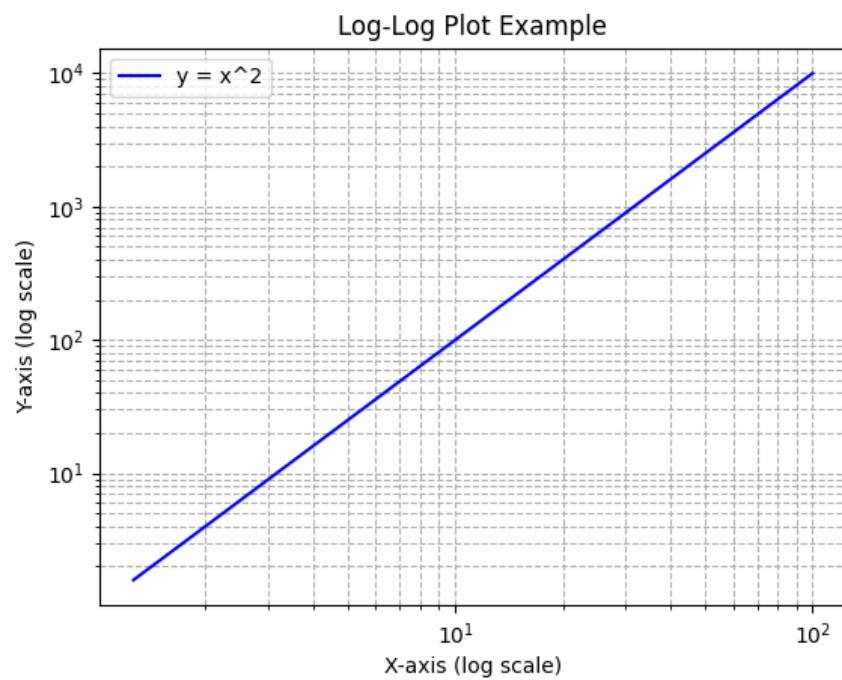
## ۲-۸-۴-۱۹) ترسیم نمودارهای *log-log* با استفاده از تابع *loglog*

python example - matplotlib\_module.py

```
117 # Matplotlib in Python
118
119 import matplotlib.pyplot as plt
120 import numpy as np
121
122 داده‌های نمونه # تولید داده‌های لگاریتمی
123 x = np.logspace(0.1, 2, 100) # تابع y = x^2
124 y = x**2
125
126 # رسم نمودار log-log
127 plt.loglog(x, y, label='y = x^2', color='blue')
128
129 # افزودن برچسب‌ها و عنوان
130 plt.xlabel('X-axis (log scale)')
131 plt.ylabel('Y-axis (log scale)')
132 plt.title('Log-Log Plot Example')
133 plt.legend()
134
135 # نمایش نمودار
136 plt.grid(True, which="both", linestyle="--")
137 plt.show()
```

شکل ۱۹-۱۵-۱۹) مثال مربوط به ترسیم نمودارهای *log-log* با استفاده از تابع *loglog*

خروجی شکل ۱۹-۱۹:



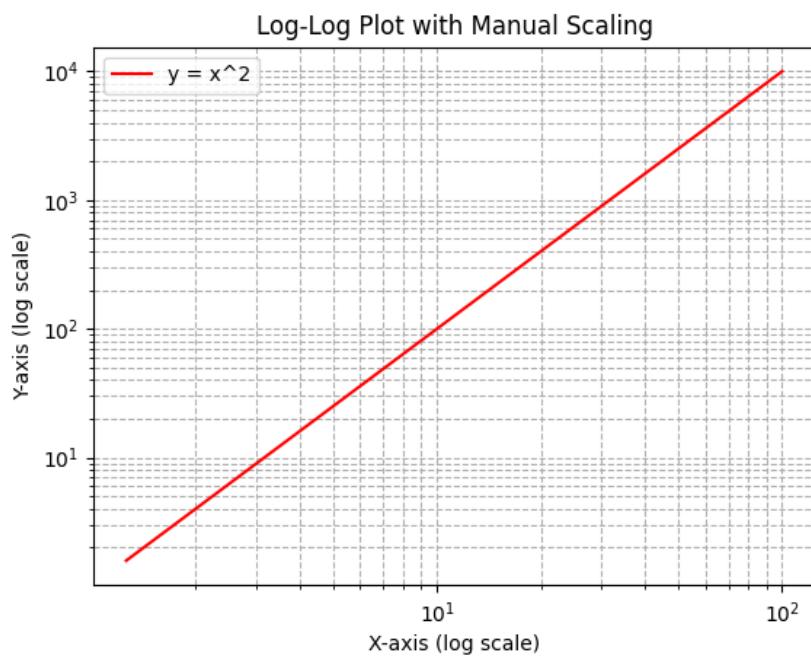
شکل ۱۹-۱۶- خروجی مربوط به ترسیم نمودارهای *log-log* با استفاده از تابع *loglog* (شکل ۱۹-۱۵)

### ۳-۸-۴-۱۹) ترسیم نمودارهای *log-log* با تنظیم دستی مقیاس محورها

python example - matplotlib\_module.py

```
141 # Matplotlib in Python
142
143 import matplotlib.pyplot as plt
144 import numpy as np
145
146 # داده‌های نمونه
147 x = np.logspace(0.1, 2, 100) # تولید داده‌های لگاریتمی
148 y = x**2 # تابع  $y = x^2$ 
149
150 # تغییر مقیاس محورها به صورت دستی
151 plt.plot(x, y, label=' $y = x^2$ ', color='red')
152 plt.xscale('log')
153 plt.yscale('log')
154
155 # افزودن جزئیات
156 plt.xlabel('X-axis (log scale)')
157 plt.ylabel('Y-axis (log scale)')
158 plt.title('Log-Log Plot with Manual Scaling')
159 plt.legend()
160
161 # نمایش نمودار
162 plt.grid(True, which="both", linestyle="--")
163 plt.show()
```

شکل ۱۹-۱۷- مثال مربوط به ترسیم نمودارهای *log-log* با تنظیم دستی مقیاس محورها



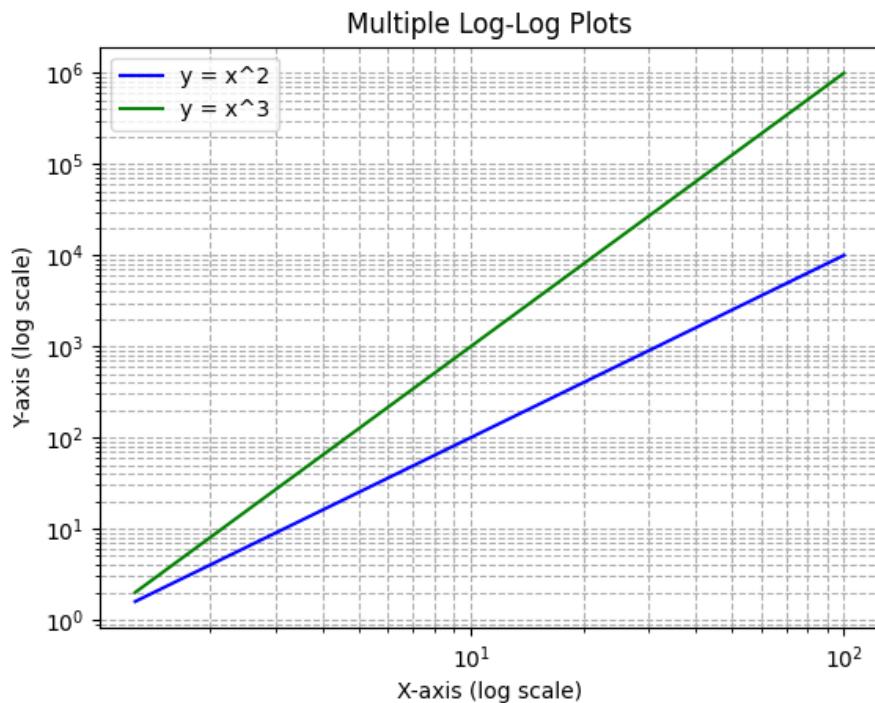
شکل ۱۸-۱۹- خروجی مربوط به ترسیم نمودارهای *log-log* با تنظیم دستی مقیاس محورها (شکل ۱۸-۱۹)

#### ۴-۸-۴-۱۹) ترسیم نمودارهای *log-log* با ترکیب چند نمودار در *log-log*

python example - matplotlib\_module.py

```
167 # Matplotlib in Python
168
169 import matplotlib.pyplot as plt
170 import numpy as np
171
172 داده‌های نمونه # داده‌های لگاریتمی
173 x = np.logspace(0.1, 2, 100) # تولید داده‌های لگاریتمی
174
175 داده‌های چند رابطه # رسم چند نمودار
176 y1 = x**2
177 y2 = x**3
178
179 # افزودن جزئیات
180 plt.xlabel('X-axis (log scale)')
181 plt.ylabel('Y-axis (log scale)')
182 plt.title('Multiple Log-Log Plots')
183 plt.legend()
184
185 # نمایش نمودار
186 plt.grid(True, which="both", linestyle="--")
187 plt.show()
```

شکل ۱۹-۱۹- مثال مربوط به ترسیم نمودارهای *log-log* با ترکیب چند نمودار در *log-log*



شکل ۱۹-۲۰- خروجی مربوط به ترسیم نمودارهای *log-log* با ترکیب چند نمودار در *log-log* (شکل ۱۹-۱۹)

#### ۱۹-۴-۸-۵) ویژگی‌های اضافی

- شبکه‌بندی دقیق‌تر: `plt.grid(which="both", linestyle="--", linewidth=0.5)`
- فقط خطوط اصلی: `which="major"`
- خطوط اصلی و فرعی: `which="both"`
- تنظیم بازه محورها: `plt.xlim(1, 100)` ○  
`plt.ylim(10, 10000)` ○
- تغییر پایه لگاریتم: `plt.xscale("log")` ○  
`plt.yscale("log")` ○

---

`plt.loglog(x, y, base=2) # مقیاس لگاریتمی با پایه ۲`

---

### ۹-۴-۱۹) ترسیم نمودارهای **Semi-log** در ماژول **matplotlib**

نمودارهای **Semi-log** در زمانی استفاده می‌شوند که یکی از محورها (افقی یا عمودی) نیاز به نمایش در مقیاس لگاریتمی دارد، در حالی که محور دیگر در مقیاس خطی باقی می‌ماند. این نمودارها برای داده‌هایی که به صورت نمایی یا لگاریتمی تغییر می‌کنند بسیار مفید هستند.

### ۱-۹-۴-۱۹) انواع نمودارهای **Semi-log**

۱. نمودار: **Semi-log-x**: محور x در مقیاس لگاریتمی و محور y در مقیاس خطی است.
۲. نمودار: **Semi-log-y**: محور y در مقیاس لگاریتمی و محور x در مقیاس خطی است.

### ۲-۹-۴-۱۹) ترسیم نمودارهای **Semi-log** در **Matplotlib**

برای رسم نمودارهای **Semi-log** می‌توان از توابع زیر استفاده کرد:

- `semilogx()`: محور x را به مقیاس لگاریتمی تنظیم می‌کند.
- `semilogy()`: محور y را به مقیاس لگاریتمی تنظیم می‌کند.

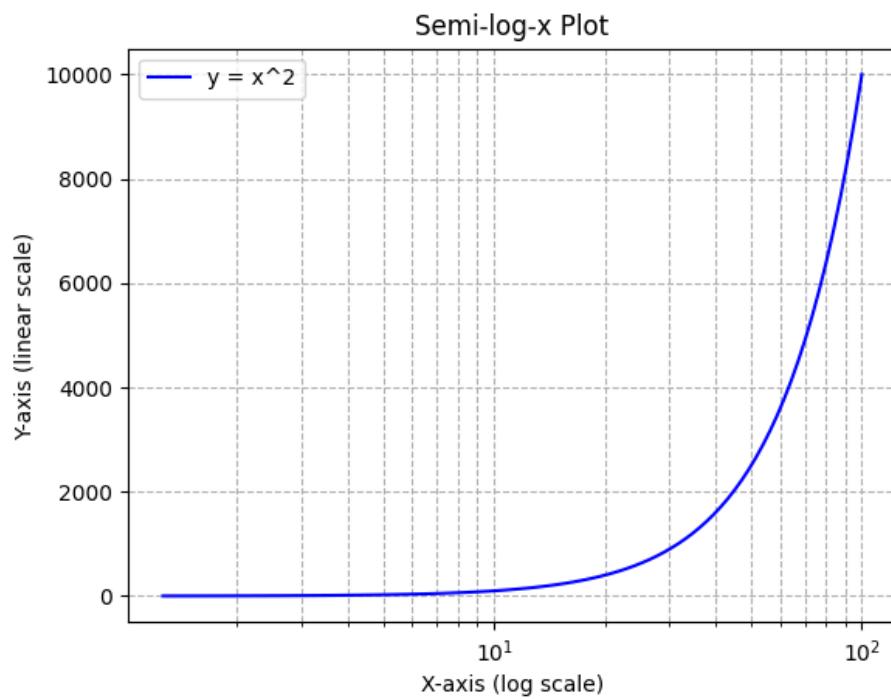
### ۳-۹-۴-۱۹) ترسیم نمودارهای Semi-log با استفاده از تابع

python example - matplotlib\_module.py

```
195 # Matplotlib in Python
196
197 import matplotlib.pyplot as plt
198 import numpy as np
199
200 #دادههای نمونه
201 x = np.logspace(0.1, 2, 100) # مقادیر لگاریتمی برای محور x
202 y = x**2 # تابع y = x^2
203
204 #رسم نمودار Semi-log-x
205 plt.semilogx(x, y, label='y = x^2', color='blue')
206
207 #افزودن جزئیات
208 plt.xlabel('X-axis (log scale)')
209 plt.ylabel('Y-axis (linear scale)')
210 plt.title('Semi-log-x Plot')
211 plt.legend()
212 plt.grid(True, which="both", linestyle="--")
213 plt.show()
214
```

شکل ۲۱-۱۹-۲۱- مثال مربوط به ترسیم نمودارهای Semi-log با استفاده از تابع  $x$

خروجی شکل ۲۱-۱۹:



شکل ۱۹-۲۲- خروجی مربوط به ترسیم نمودارهای *Semi-log* با استفاده از تابع  $y = x^2$  (شکل ۱۹-۲۱)

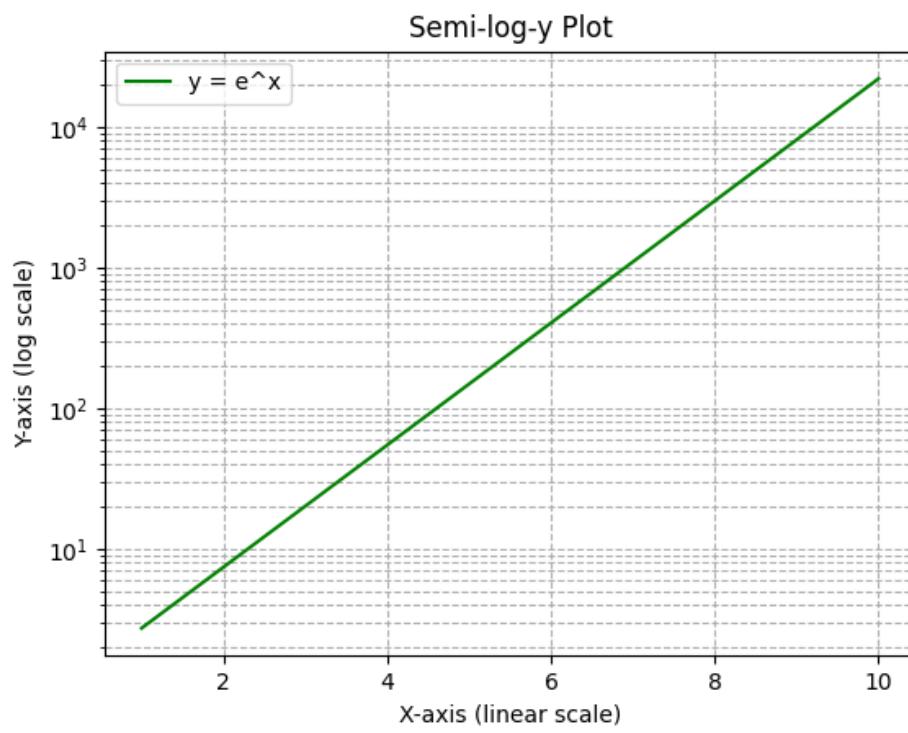
## ۴-۹-۴-۱۹) ترسیم نمودارهای Semi-log با استفاده از تابع Semi-log-y

python example - matplotlib\_module.py

```
217 # Matplotlib in Python
218
219 import matplotlib.pyplot as plt
220 import numpy as np
221
222 # داده‌های نمونه
223 x = np.linspace(1, 10, 100) # مقادیر خطی برای محور x
224
225 y = np.exp(x) # تابع y = e^x
226
227 # رسم نمودار Semi-log-y
228 plt.semilogy(x, y, label='y = e^x', color='green')
229
230 # افزودن جزئیات
231 plt.xlabel('X-axis (linear scale)')
232 plt.ylabel('Y-axis (log scale)')
233 plt.title('Semi-log-y Plot')
234 plt.legend()
235 plt.grid(True, which="both", linestyle="--")
236 plt.show()
```

شکل ۱۹-۲۳-مثال مربوط به ترسیم نمودارهای Semi-log با استفاده از تابع Semi-log-y

خروجی شکل ۱۹-۲۳:



شکل ۱۹-۲۴- خروجی مربوط به ترسیم نمودارهای *Semi-log* با استفاده از تابع  $y = e^x$  (شکل ۱۹-۲۳)

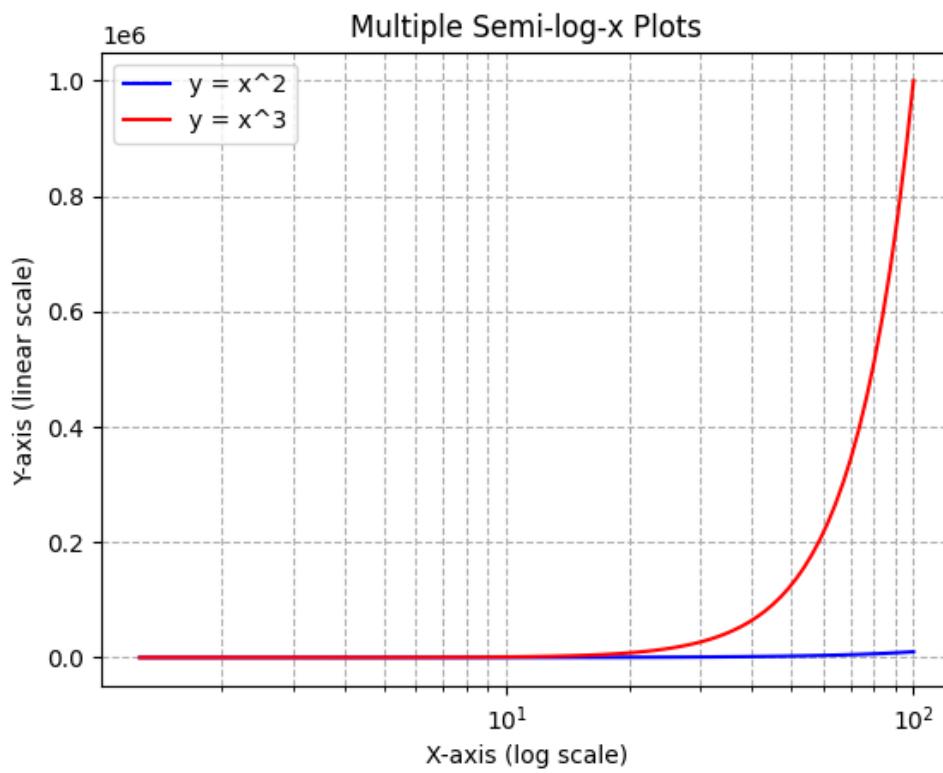
## ۱۹-۴-۵) مثال مربوط به ترسیم نمودارهای Semi-log با ترکیب چند نمودار

python example - matplotlib\_module.py

```
240 # Matplotlib in Python
241
242 import matplotlib.pyplot as plt
243 import numpy as np
244
245 داده‌های نمونه #
246 x = np.logspace(0.1, 2, 100)
247 y1 = x**2
248 y2 = x**3
249
250 رسم دو نمودار #
251 plt.semilogx(x, y1, label='y = x^2', color='blue')
252 plt.semilogx(x, y2, label='y = x^3', color='red')
253
254 افزودن جزئیات #
255 plt.xlabel('X-axis (log scale)')
256 plt.ylabel('Y-axis (linear scale)')
257 plt.title('Multiple Semi-log-x Plots')
258 plt.legend()
259 plt.grid(True, which="both", linestyle="--")
260 plt.show()
```

شکل ۱۹-۲۵-۱۹) مثال مربوط به ترسیم نمودارهای Semi-log با ترکیب چند نمودار

خروجی شکل ۱۹-۲۵:



شکل ۱۹-۲۶- خروجی مربوط به ترسیم نمودارهای *Semi-log* با ترکیب چند نمودار *Semi-log* (شکل ۱۹-۲۵)

#### ۱۹-۴-۶) ویژگی‌های قابل تنظیم

- شبکه‌بندی دقیق‌تر:

```
plt.grid(which="both", linestyle="--", linewidth=0.5) ○
```

- تنظیم بازه محورها:

```
plt.xlim(1, 100) ○
```

```
plt.ylim(10, 10000) ○
```

- تغییر پایه لگاریتم:

---

```
plt.semilogx(x, y, base=2) # x پایه لگاریتمی ۲ برای محور
```

---

## ۵-۱۹) محورها و برچسبها در مازول **Matplotlib**

در **Matplotlib**, محورها<sup>۱۲۴</sup> و برچسبها<sup>۱۲۵</sup> نقش مهمی در نمایش و توضیح نمودارها دارند. این عناصر به مخاطب کمک می‌کنند تا داده‌ها و روابط آن‌ها را بهتر درک کند. در ادامه، روش‌های مختلف برای مدیریت محورها و برچسبها در نمودارها توضیح داده می‌شوند.

### ۱-۵-۱۹) برچسب‌گذاری محورها

برای نمایش توضیحات محورهای افقی<sup>۱۲۶</sup> و عمودی<sup>۱۲۷</sup> از توابع زیر استفاده می‌شود:

• plt.xlabel(label, \*\*kwargs): برای برچسب‌گذاری محور افقی.

• plt.ylabel(label, \*\*kwargs): برای برچسب‌گذاری محور عمودی.

مثال:

---

<sup>124</sup> Axes

<sup>125</sup> Labels

<sup>126</sup> X-axis

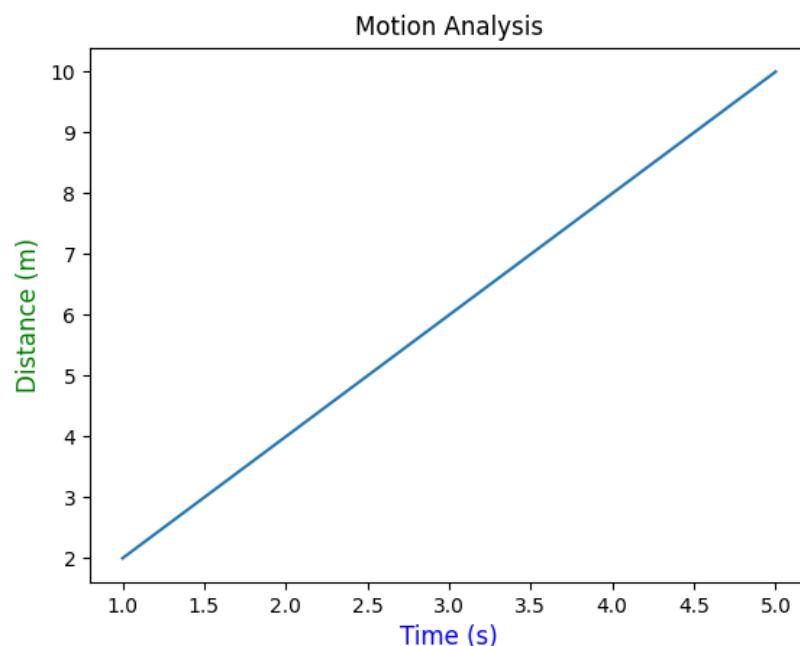
<sup>127</sup> Y-axis

python example - matplotlib\_module.py

```
264 # Matplotlib in Python  
265  
266 import matplotlib.pyplot as plt  
267  
268 x = [1, 2, 3, 4, 5]  
269 y = [2, 4, 6, 8, 10]  
270  
271 plt.plot(x, y)  
272 plt.xlabel('Time (s)', fontsize=12, color='blue') # برچسب محور افقی  
273 plt.ylabel('Distance (m)', fontsize=12, color='green') # برچسب محور عمودی  
274 plt.title('Motion Analysis') # عنوان نمودار  
275 plt.show()
```

شکل ۱۹-۲۷- مثال مربوط به انواع برچسب‌گذاری در نمودارهای *matplotlib*

خروجی شکل ۱۹-۲۷:



شکل ۱۹-۲۸- خروجی مربوط به انواع برچسب‌گذاری در *matplotlib* (شکل ۱۹-۲۷)

## ۲-۵-۱۹) تنظیم محدوده محورها

• plt.xlim([min, max]): تنظیم محدوده محور افقی.

• plt.ylim([min, max]): تنظیم محدوده محور عمودی.

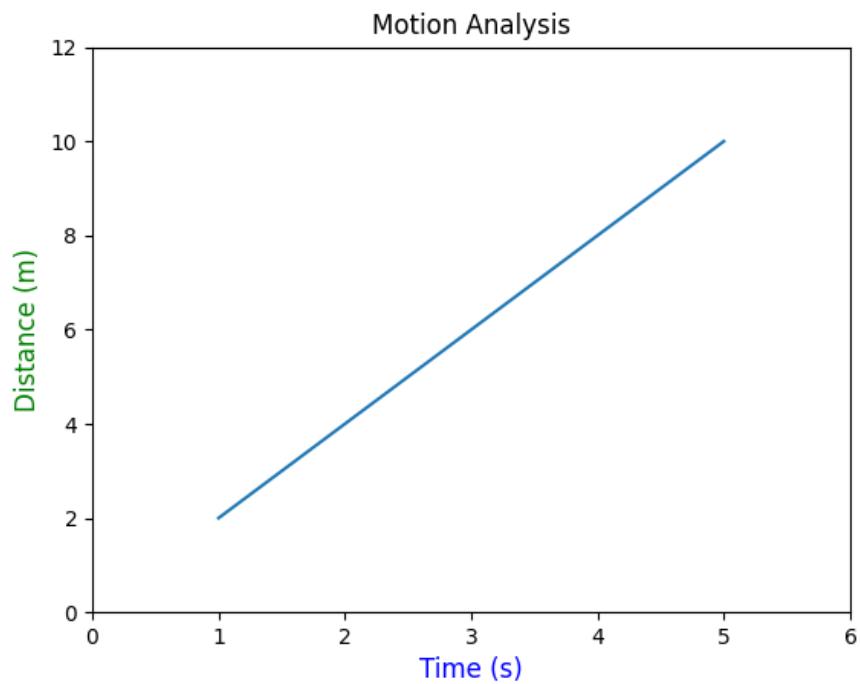
مثال:

python example - matplotlib\_module.py

```
279 # Matplotlib in Python
280 import matplotlib.pyplot as plt
281
282 x = [1, 2, 3, 4, 5]
283 y = [2, 4, 6, 8, 10]
284
285 plt.plot(x, y)
286 plt.xlabel('Time (s)', fontsize=12, color='blue') # برچسب محور افقی
287 plt.ylabel('Distance (m)', fontsize=12, color='green') # برچسب محور عمودی
288 plt.title('Motion Analysis') # عنوان نمودار
289 plt.xlim(0, 6) # X محدوده محور
290 plt.ylim(0, 12) # Y محدوده محور
291 plt.show()
```

شکل ۲۹-۱۹- مثال مربوط به تنظیم محدوده محورها در *matplotlib*

خروجی شکل ۲۹-۱۹:



شکل ۱۹-۳۰- خروجی مربوط به تنظیم محدوده محورها (شکل ۲۹-۱۹)

۳-۵-۱۹) اضافه کردن شبکه به محورها

- plt.grid(True): اضافه کردن خطوط شبکه به نمودار

- plt.grid(axis='x'): نمایش خطوط شبکه فقط برای محور X

- plt.grid(axis='y'): نمایش خطوط شبکه فقط برای محور Y

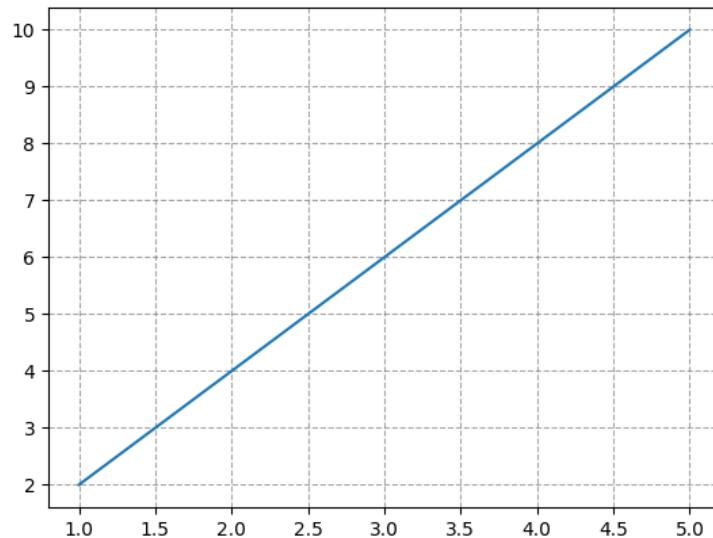
مثال:

python example - matplotlib\_module.py

```
295 # Matplotlib in Python  
296 import matplotlib.pyplot as plt  
297  
298 x = [1, 2, 3, 4, 5]  
299 y = [2, 4, 6, 8, 10]  
300  
301 plt.plot(x, y)  
302 plt.grid(True, linestyle='--', color='gray', alpha=0.7)  
303 plt.show()
```

شکل ۱۹-۳۱- مثالی از افزودن شبکه به محورها در *matplotlib*

خروجی شکل ۱۹-۳۱:



شکل ۱۹-۳۲- خروجی مربوط به افزودن شبکه به محورها (شکل ۱۹-۳۱)

#### ۴-۵-۱۹) برچسب‌های دلخواه برای محورها

برای تغییر مقادیر پیش‌فرض نمایش داده شده روی محورها:

• plt.xticks(ticks, labels, \*\*kwargs): تنظیم مقادیر و برچسب‌های محور افقی.

• plt.yticks(ticks, labels, \*\*kwargs): تنظیم مقادیر و برچسب‌های محور عمودی.

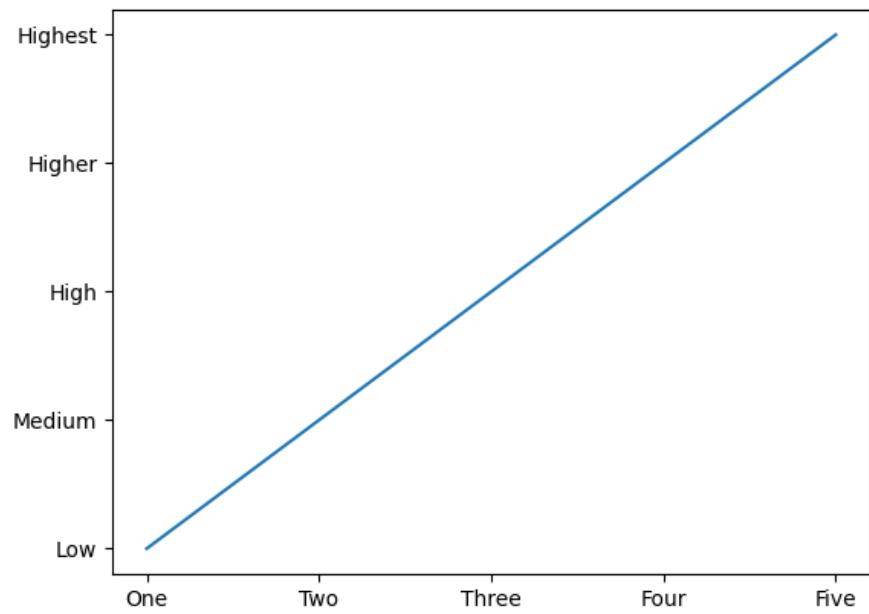
مثال:

python example - matplotlib\_module.py

```
307 # Matplotlib in Python
308 import matplotlib.pyplot as plt
309
310 x = [1, 2, 3, 4, 5]
311 y = [2, 4, 6, 8, 10]
312
313 ticks_x = [1, 2, 3, 4, 5]
314 labels_x = ['One', 'Two', 'Three', 'Four', 'Five']
315
316 ticks_y = [2, 4, 6, 8, 10]
317 labels_y = ['Low', 'Medium', 'High', 'Higher', 'Highest']
318
319 plt.plot(x, y)
320 plt.xticks(ticks_x, labels_x) # X تغییر مقادیر محور
321 plt.yticks(ticks_y, labels_y) # Y تغییر مقادیر محور
322 plt.show()
```

شكل ۱۹-۳۳- مثال مربوط به برچسب‌گذاری دلخواه برای محورها

خروجی شکل ۱۹-۳۳:



شکل ۱۹-۳۴- خروجی مربوط به برچسب‌گذاری دلخواه برای محورها (شکل ۱۹-۳۳)

#### ۱۹-۵-۵) چرخش برچسب‌های محورها

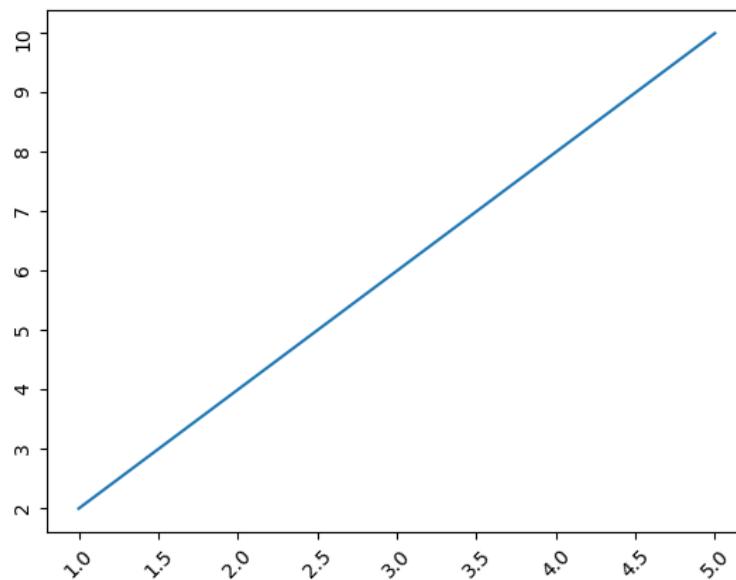
برای چرخاندن برچسب‌های محور، از آرگومان rotation استفاده می‌شود:

python example - matplotlib\_module.py

```
326 # Matplotlib in Python  
327 import matplotlib.pyplot as plt  
328  
329 x = [1, 2, 3, 4, 5]  
330 y = [2, 4, 6, 8, 10]  
331  
332 plt.plot(x, y)  
333 plt.xticks(rotation=45) # چرخش برچسب‌های محور X  
334 plt.yticks(rotation=90) # چرخش برچسب‌های محور Y  
335 plt.show()
```

شکل ۱۹-۳۵- مثال مربوط به چرخش برچسب‌های محورها

خروجی شکل ۱۹-۳۵:



شکل ۱۹-۳۶- خروجی مربوط به چرخش برچسب‌های محورها (شکل ۱۹-۳۵)

## ۱۹-۵-۶) افزودن خطوط به محورها

• رسم یک خط افقی روی نمودار:`plt.axhline()`

• رسم یک خط عمودی روی نمودار:`plt.axvline()`

• افزودن یک ناحیه افقی:`plt.axhspan()`

• افزودن یک ناحیه عمودی:`plt.axvspan()`

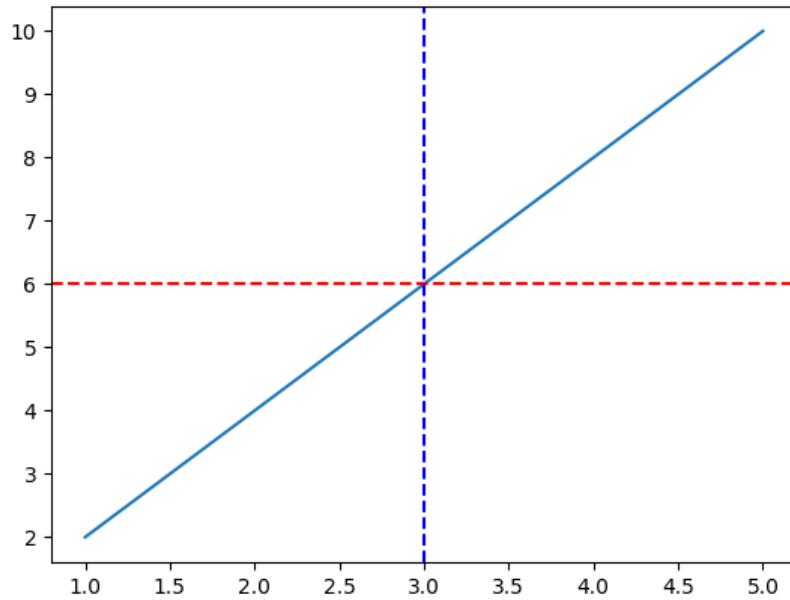
مثال:

python example - matplotlib\_module.py

```
339 # Matplotlib in Python
340 import matplotlib.pyplot as plt
341
342 x = [1, 2, 3, 4, 5]
343 y = [2, 4, 6, 8, 10]
344
345 plt.plot(x, y)
346 plt.axhline(y=6, color='red', linestyle='--') # خط افقی
347 plt.axvline(x=3, color='blue', linestyle='--') # خط عمودی
348 plt.show()
```

شکل ۱۹-۳۷- مثال مربوط به افزودن خطوط به محورها

خروجی شکل ۱۹-۳۷:



شکل ۱۹-۳۸- خروجی مربوط به افزودن خطوط به محورها (شکل ۱۹-۳۷)

#### ۶-۱۹) ایجاد چندین نمودار در یک صفحه

در تابع `subplot()` برای ایجاد چندین نمودار در یک صفحه استفاده می‌شود. این قابلیت به کاربران اجازه می‌دهد داده‌های مختلف را در یک قالب مشترک مقایسه و تحلیل کنند. هر نمودار در یک شبکه<sup>۱۲۸</sup> قرار می‌گیرد و می‌توان آن‌ها را به صورت مرتب و سازماندهی شده نمایش داد.

#### ۱-۶-۱۹) ساختار کلی `subplot()`

تابع `( subplot()` سه آرگومان می‌گیرد که تعیین می‌کنند نمودارها چگونه در یک صفحه سازماندهی شوند:

<sup>128</sup> Grid

---

### *plt.subplot(nrows, ncols, index)*

---

• *nrows*: تعداد ردیف‌های نمودارها

• *ncols*: تعداد ستون‌های نمودارها

• *index*: شماره نمودار موردنظر (از بالا-چپ شروع می‌شود و از ۱ شروع می‌کند).

## ۱۹-۶-۲) رسم چند نمودار ساده

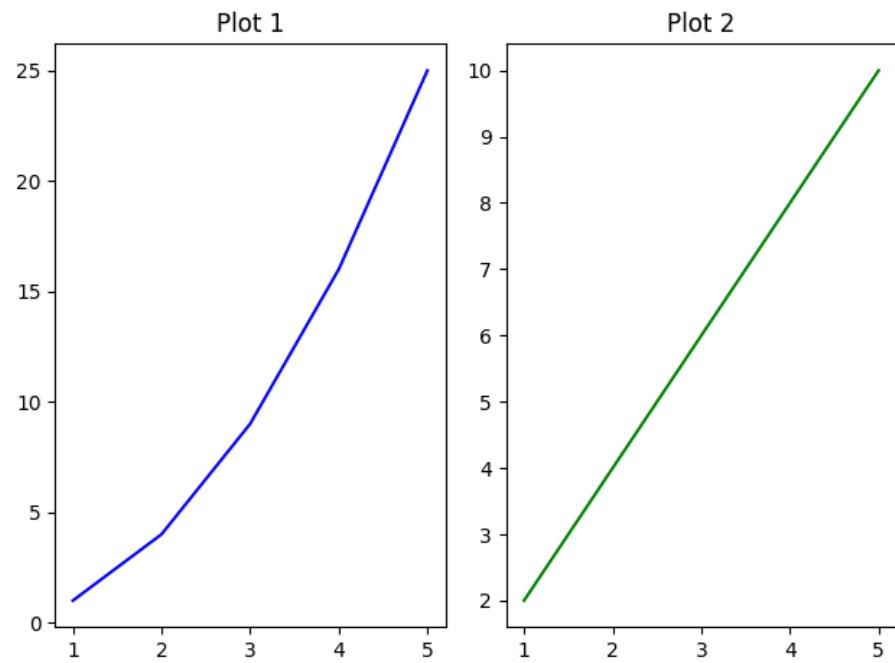
مثال ترسیم دو نمودار در یک ردیف و دو ستون:

python example - matplotlib\_module.py

```
365 # Matplotlib in Python
366 import matplotlib.pyplot as plt
367
368 x = [1, 2, 3, 4, 5]
369 y1 = [1, 4, 9, 16, 25]
370 y2 = [2, 4, 6, 8, 10]
371
372 plt.subplot(1, 2, 1) # نمودار اول
373 plt.plot(x, y1, color='blue')
374 plt.title('Plot 1')
375
376 plt.subplot(1, 2, 2) # نمودار دوم
377 plt.plot(x, y2, color='green')
378 plt.title('Plot 2')
379
380 plt.tight_layout() # تنظیم فاصله‌ها بین نمودارها
381 plt.show()
```

شکل ۱۹-۳۹-۳۹) مثال مربوط به ترسیم دو نمودار در یک ردیف و دو ستون

خروجی شکل ۱۹-۳۹



شکل ۱۹-۴۰- خروجی مربوط به ترسیم دو نمودار در یک ردیف و دو ستون (شکل ۱۹-۳۹)

۳-۶-۱۹) چند نمودار در چند ردیف و ستون

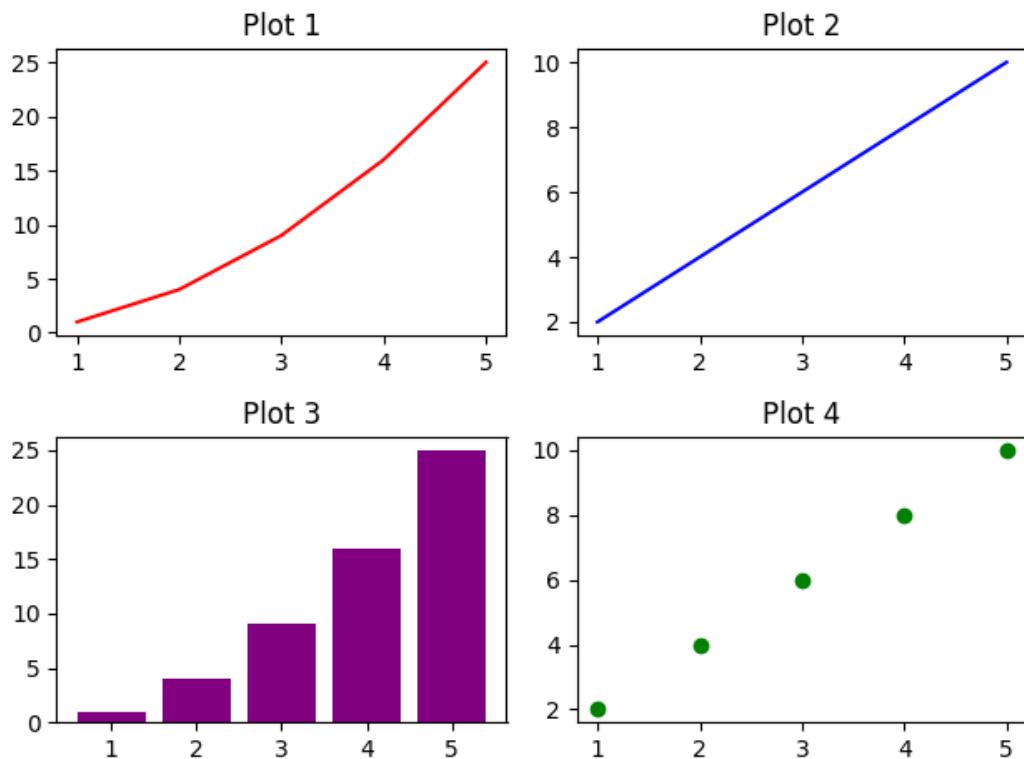
مثال ترسیم چهار نمودار در دو ردیف و دو ستون:

python example - matplotlib\_module.py

```
385 # Matplotlib in Python
386 import matplotlib.pyplot as plt
387
388 x = [1, 2, 3, 4, 5]
389 y1 = [1, 4, 9, 16, 25]
390 y2 = [2, 4, 6, 8, 10]
391
392 # دو ردیف، دو ستون، نمودار اول
393 plt.subplot(2, 2, 1)
394 plt.plot(x, y1, color='red')
395 plt.title('Plot 1')
396
397 # دو ردیف، دو ستون، نمودار دوم
398 plt.subplot(2, 2, 2)
399 plt.plot(x, y2, color='blue')
400 plt.title('Plot 2')
401
402 # دو ردیف، دو ستون، نمودار سوم
403 plt.subplot(2, 2, 3)
404 plt.bar(x, y1, color='purple')
405 plt.title('Plot 3')
406
407 # دو ردیف، دو ستون، نمودار چهارم
408 plt.subplot(2, 2, 4)
409 plt.scatter(x, y2, color='green')
410 plt.title('Plot 4')
411
412 # تنظیم خودکار فاصله‌ها
413 plt.tight_layout()
414
415 plt.show()
```

شکل ۱۹-۴۱- مثال مربوط به ترسیم چهار نمودار در دو ردیف و دو ستون

خروجی شکل ۱۹-۴۱:



شکل ۱۹-۴۲- خروجی مربوط به ترسیم چهار نمودار در دو ردیف و دو ستون (شکل ۱۹-۴۱)

#### ۱۹-۶-۴) استفاده از **Figure** و **Axes** برای Subplot

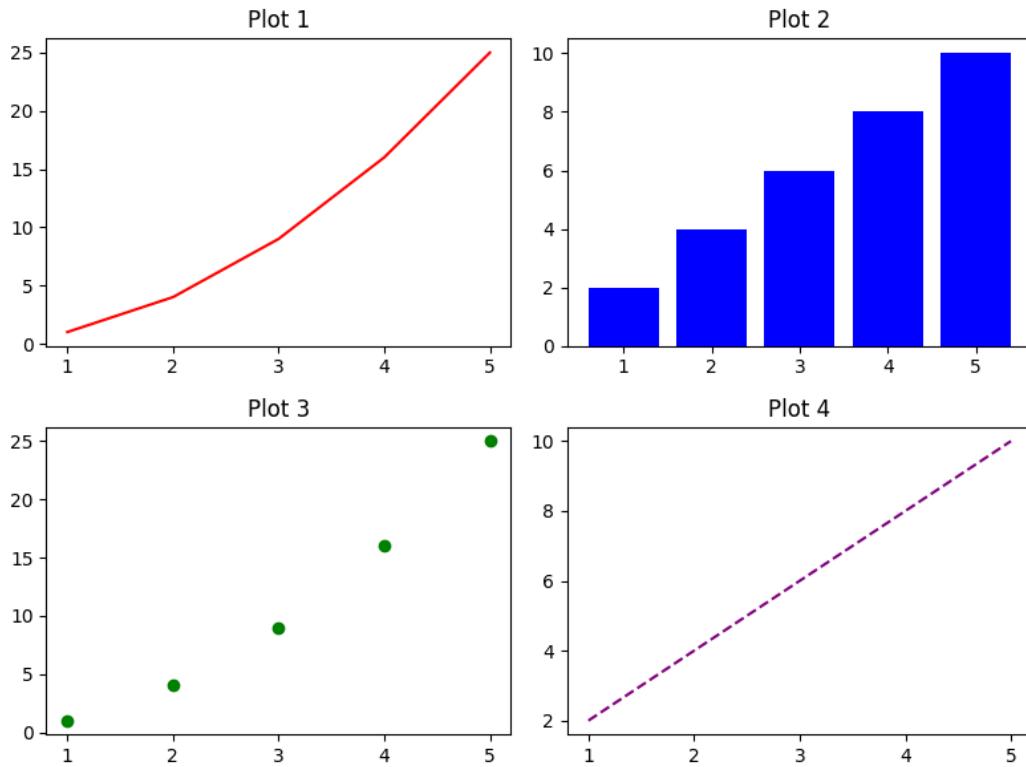
در `Matplotlib`, می‌توان از روش پیشرفته‌تر `plt.subplots()` برای ایجاد چند نمودار استفاده کرد. این روش کنترل بیشتری روی نمودارها فراهم می‌کند.

python example - matplotlib\_module.py

```
413 # Matplotlib in Python
414 import matplotlib.pyplot as plt
415
416 x = [1, 2, 3, 4, 5]
417 y1 = [1, 4, 9, 16, 25]
418 y2 = [2, 4, 6, 8, 10]
419
420 fig, axes = plt.subplots(2, 2, figsize=(8, 6)) # دو ردیف، دو ستون
421
422 axes[0, 0].plot(x, y1, color='red')
423 axes[0, 0].set_title('Plot 1')
424
425 axes[0, 1].bar(x, y2, color='blue')
426 axes[0, 1].set_title('Plot 2')
427
428 axes[1, 0].scatter(x, y1, color='green')
429 axes[1, 0].set_title('Plot 3')
430
431 axes[1, 1].plot(x, y2, linestyle='--', color='purple')
432 axes[1, 1].set_title('Plot 4')
433
434 plt.tight_layout()
435 plt.show()
```

شکل ۱۹-۴۳- مثال مربوط به روش دوم برای ترسیم چند نمودار در یک صفحه

خروجی شکل ۱۹-۴۳:



شکل ۱۹-۴۴- خروجی مربوط به روش دوم برای ترسیم چند نمودار در یک صفحه (شکل ۱۹-۴۳)

## ۷-۱۹) نمودارهای هیستوگرام و میله‌ای در ماژول **Matplotlib**

در Matplotlib، هر دو نمودار هیستوگرام و میله‌ای برای نمایش داده‌ها استفاده می‌شوند، اما کاربرد و نحوه نمایش آن‌ها متفاوت است.

### ۱-۷-۱۹) نمودار هیستوگرام

هیستوگرام برای نمایش توزیع فراوانی داده‌های عددی به کار می‌رود. داده‌ها به بازه‌های پیوسته تقسیم شده و تعداد داده‌های موجود در هر بازه نشان داده می‌شود.

### ۱-۷-۱۹) ویژگی‌ها

- مناسب برای داده‌های عددی و پیوسته.
- نشان‌دهنده چگونگی توزیع داده‌ها.
- محور X نشان‌دهنده بازه‌ها و محور Y نشان‌دهنده تعداد یا فراوانی است.

### ۲-۱-۷-۱۹) توابع اصلی نمودار هیستوگرام

- plt.hist(): برای رسم هیستوگرام.

### ۳-۱-۷-۱۹) آرگومان‌های نمودار هیستوگرام

- bins: تعداد یا بازه‌های دسته‌بندی
- color: رنگ میله‌ها.
- alpha: شفافیت میله‌ها.
- density: نرمال‌سازی فراوانی (برای نمایش چگالی).

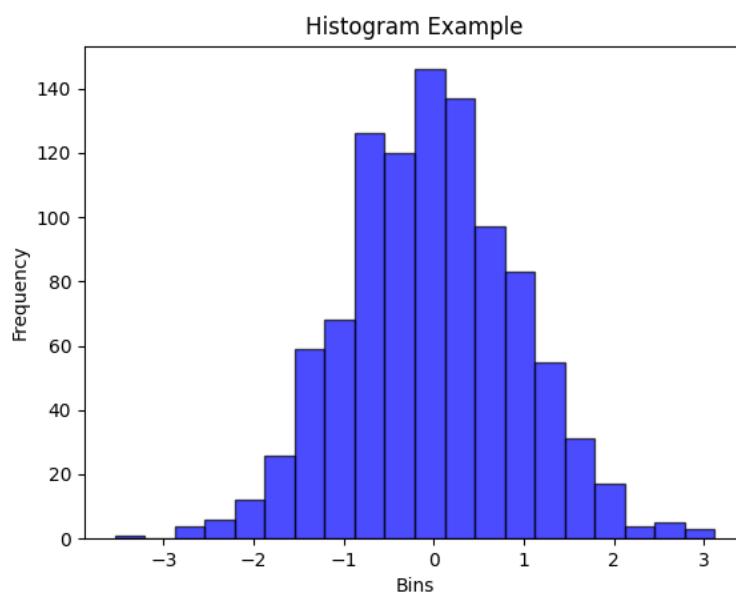
#### ۱۹-۷-۴) مثال برای نمودار هیستوگرام

python example - matplotlib\_module.py

```
439 # Matplotlib in Python
440 import matplotlib.pyplot as plt
441 import numpy as np
442
443 data = np.random.randn(1000) # تولید داده‌های تصادفی
444 plt.hist(data, bins=20, color='blue', alpha=0.7, edgecolor='black')
445 plt.title('Histogram Example')
446 plt.xlabel('Bins')
447 plt.ylabel('Frequency')
448 plt.show()
```

شکل ۱۹-۴۵-۴۵) مثال مربوط به نمودار هیستوگرام

خروجی شکل ۱۹-۴۵



شکل ۱۹-۴۶- خروجی مربوط به نمودار هیستوگرام (شکل ۱۹-۴۵)

### ۲-۷-۱۹) نمودار میله‌ای

نمودار میله‌ای برای نمایش داده‌های دسته‌بندی شده استفاده می‌شود. این نمودار مقادیر یا فراوانی هر دسته را با میله‌هایی جداگانه نشان می‌دهد.

### ۱-۲-۷-۱۹) ویژگی‌های نمودار میله‌ای

- مناسب برای داده‌های گستته و دسته‌بندی شده.
- محور X نشان‌دهنده دسته‌ها و محور Y نشان‌دهنده مقادیر یا فراوانی است.
- فاصله‌ای مشخص بین میله‌ها وجود دارد.

### ۲-۲-۷-۱۹) توابع اصلی نمودار میله‌ای

plt.bar(): برای رسم میله‌ها به صورت عمودی •

plt.barch(): برای رسم میله‌ها به صورت افقی •

### ۳-۲-۷-۱۹) آرگومان‌های نمودار میله‌ای

color: رنگ میله‌ها. •

width: پهنای میله‌ها. •

align: موقعیت میله‌ها روی محور center یا edge. •

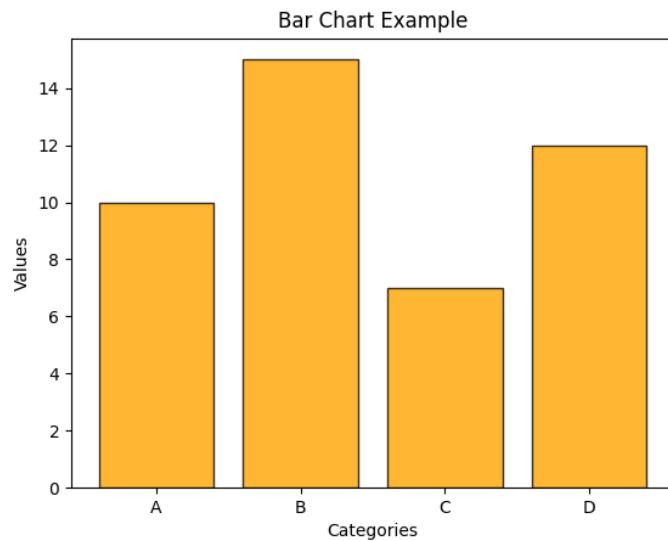
## ۴-۲-۷-۱۹) مثال برای نمودار میله‌ای

python example - matplotlib\_module.py

```
452 # Matplotlib in Python
453 import matplotlib.pyplot as plt
454
455 categories = ['A', 'B', 'C', 'D']
456 values = [10, 15, 7, 12]
457
458 plt.bar(categories, values, color='orange', alpha=0.8, edgecolor='black')
459 plt.title('Bar Chart Example')
460 plt.xlabel('Categories')
461 plt.ylabel('Values')
462 plt.show()
```

شکل ۱۹-۴۷- مثال مربوط به نمودار میله‌ای

خروجی ۴۷-۱۹:



شکل ۱۹-۴۸- خروجی مربوط به نمودار میله‌ای (شکل ۱۹-۴۷)

### ۳-۷-۱۹) تفاوت‌های اصلی بین هیستوگرام و میله‌ای

جدول ۱۹-۱- جدول مربوط به تفاوت‌های نمودارهای هیستوگرام و میله‌ای

| نمودار میله‌ای          | نمودار هیستوگرام       | ویژگی         |
|-------------------------|------------------------|---------------|
| داده‌های گسسته          | داده‌های پیوسته        | نوع داده      |
| دسته‌ها                 | بازه‌ها                | X محور        |
| میله‌ها از هم جدا هستند | میله‌ها به هم متصل‌اند | فاصله میله‌ها |
| مقایسه مقدار دسته‌ها    | تحلیل توزیع فراوانی    | کاربرد        |

### ۴-۷-۱۹) ترکیب نمودارهای هیستوگرام و میله‌ای

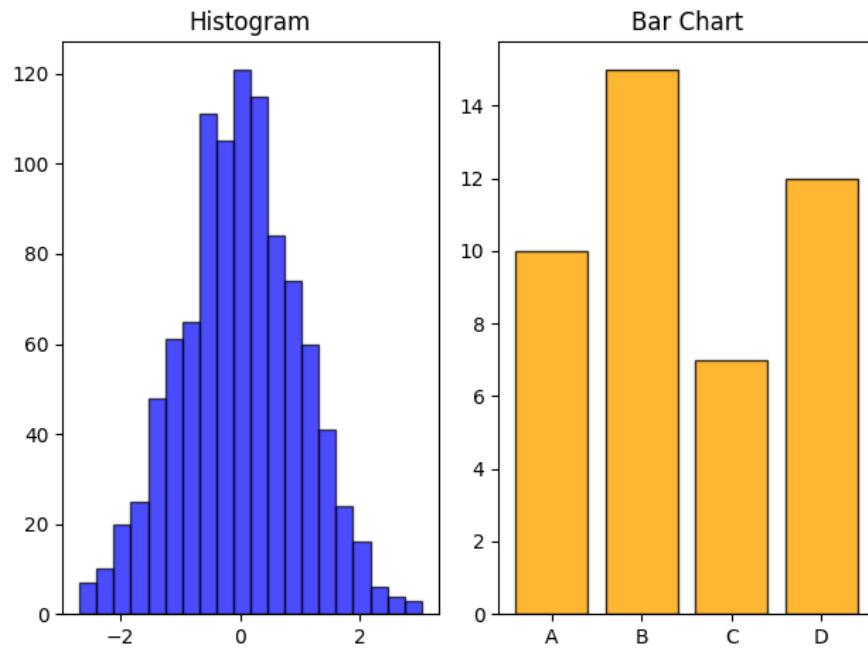
در برخی موارد، می‌توان از هر دو نمودار برای تحلیل داده‌ها در یک صفحه استفاده کرد:

python example - matplotlib\_module.py

```
466 # Matplotlib in Python
467 import matplotlib.pyplot as plt
468 import numpy as np
469
470 data = np.random.randn(1000)
471 categories = ['A', 'B', 'C', 'D']
472 values = [10, 15, 7, 12]
473
474 plt.subplot(1, 2, 1) # رسم هیستوگرام
475 plt.hist(data, bins=20, color='blue', alpha=0.7, edgecolor='black')
476 plt.title('Histogram')
477
478 plt.subplot(1, 2, 2) # رسم نمودار میله‌ای
479 plt.bar(categories, values, color='orange', alpha=0.8, edgecolor='black')
480 plt.title('Bar Chart')
481
482 plt.tight_layout()
483 plt.show()
```

شکل ۱۹-۴۹- مثال مربوط به ترکیب نمودارهای هیستوگرام و میله‌ای

خروجی شکل ۱۹-۴۹:



شکل ۱۹-۵۰- خروجی مربوط به ترکیب نمودارهای هیستوگرام و میله‌ای (شکل ۱۹-۴۹)

نکات:

- هیستوگرام برای تحلیل توزیع داده‌های پیوسته به کار می‌رود.
- نمودار میله‌ای برای مقایسه داده‌های دسته‌بندی شده استفاده می‌شود. هر دو ابزار قدرتمندی برای تحلیل و نمایش داده‌ها در **Matplotlib** هستند.

## منابع

1. Gowrishankar, S., & Veena, A. (2018). Introduction to Python programming. Chapman and Hall/CRC.
2. Lutz, M. (2010). Programming python. " O'Reilly Media, Inc.".
3. Padmanabhan, T. R. (2016). Programming with python. Springer Singapore.
4. Guzdial, M. J., & Ericson, B. (2015). Introduction to computing and programming in python. Pearson.
5. Liang, Y. D. (2013). Introduction to programming using Python. Pearson.
6. Kuhlman, D. (2009). A python book: Beginning python, advanced python, and python exercises (pp. 1-227). Lutz: Dave Kuhlman.
7. Sedgewick, R., Wayne, K., & Dondero, R. (2015). Introduction to programming in Python: An interdisciplinary approach. Addison-Wesley Professional.
8. Sundnes, J. (2020). Introduction to scientific programming with Python (p. 148). Springer Nature.
9. Langtangen, H. P. (2014). A primer on scientific programming with Python (Vol. 6). Springer.
10. José, U. (2021). Python programming for data analysis. Springer Nature.
11. Mastrodomenico, R. (2022). The python book. John Wiley & Sons.
12. Kaswan, K. S., Dhatterwal, J. S., & Balamurugan, B. (2023). Python for Beginners. Chapman and Hall/CRC.
13. Haslwanter, T. (2016). An introduction to statistics with python. With applications in the life sciences. Switzerland: Springer International Publishing.
14. Kong, Q., Siauw, T., & Bayen, A. (2020). Python programming and numerical methods: A guide for engineers and scientists. Academic Press.