

بسم تعالی

آموزش مقدماتی تا پیشرفته

زبان برنامه‌نویسی پایتون

رضا سیدنژاد

دکتر مهدی رضوی‌فر



پیشگفتار

فهرست عناوین

۱. آشنایی با برنامهنویسی و زبان‌های برنامهنویسی ۲۱
۱-۱) اهمیت و نقش برنامهنویسی ۲۱
۱-۲) گام‌های اولیه در یادگیری برنامهنویسی ۲۲
۱-۳) خلاقیت در برنامهنویسی ۲۲
۱-۴) تعریف برنامهنویسی ۲۲
۱-۵) تاریخچه برنامهنویسی ۲۴
۱-۶) زبان‌های برنامهنویسی ۲۵
۱-۶-۱) آشنایی با زبان برنامهنویسی پایتون ۲۷
۱-۶-۲) آشنایی با زبان برنامهنویسی سی ۲۷
۱-۶-۳) آشنایی با زبان برنامهنویسی سی پلاس پلاس ۲۹
۱-۶-۴) آشنایی با زبان برنامهنویسی سی شارپ ۳۰
۱-۶-۵) آشنایی با زبان برنامهنویسی جاوا ۳۱
۱-۶-۶) آشنایی با زبان برنامهنویسی متلب ۳۳
۲) آشنایی با زبان برنامهنویسی پایتون و نصب و راهاندازی آن ۳۶
۲-۱) تاریخچه زبان برنامهنویسی پایتون ۳۷
۲-۲) ویژگی‌های زبان برنامهنویسی پایتون ۳۸
۲-۳) کاربردهای زبان برنامهنویسی پایتون ۴۰

۴۲.....	۴-۲) جایگاه زبان پایتون در شرکت‌های بزرگ دنیا.....
۴۳.....	۵-۲) نصب و راه اندازی پایتون.....
۵۰.....	۶-۲) آشنایی با محیط VS Code و نصب افزونه‌های پایتون
۵۴.....	۷-۲) اجرای کدهای پایتون با VS Code
۵۸.....	۳) آشنایی با مقدمات و ساختار زبان برنامه‌نویسی پایتون
۵۸.....	۱-۳) نحوه کامنت‌گذاری در پایتون.....
۶۰.....	۲-۳) آشنایی تابع چاپ (print).....
۶۲.....	۳-۳) آشنایی با f-string در پایتون
۶۳.....	۴-۳) انواع عمل‌گرها در پایتون.....
۶۴.....	۴-۳) عمل‌گرها ریاضی
۶۵.....	۴-۳) عمل‌گرها مقایسه‌ای
۶۶.....	۴-۳) عمل‌گرها تخصیص
۶۸.....	۴-۳) عمل‌گرها منطقی
۷۰.....	۴-۳) اولویت عمل‌گرها
۷۱.....	۵-۳) معرفی توابع وابسته
۷۳.....	۴) متغیرها و انواع داده‌ها در زبان برنامه‌نویسی پایتون
۷۳.....	۱-۴) متغیرها در پایتون
۷۵.....	۲-۴) انواع داده‌ها در پایتون

۷۵.....	۱-۲-۴) اعداد در پایتون
۷۹.....	۲-۲-۴) رشته‌ها در پایتون
۱۰۱.....	۳-۲-۴) لیست‌ها (فهرست‌ها) در پایتون
۱۲۱.....	۴-۲-۴) دیکشنری‌ها در پایتون
۱۴۱.....	۵-۲-۴) تاپل‌ها در پایتون
۱۵۴.....	۶-۲-۴) مجموعه‌ها در پایتون
۱۷۱.....	۷) دستورات شرطی در پایتون
۱۷۲.....	۸-۱) دستور شرطی if
۱۷۴.....	۹-۱) ساختار if
۱۷۵.....	۹-۲) ساختار if/else
۱۷۶.....	۹-۳) ساختار if/elif/else
۱۷۸.....	۹-۴) ساختار if/elif/else با بیش از دو شرط:
۱۸۳.....	۹-۵) دستورات شرطی تو در تو
۱۸۸.....	۱۰) حلقه‌ها در پایتون
۱۸۸.....	۱۱-۱) حلقه while در پایتون
۱۸۹.....	۱۱-۲) مثال‌های مربوط به حلقه while
۱۹۳.....	۱۱-۳) بخش else در حلقه while
۱۹۵.....	۱۱-۴) دستور continue و break در حلقه while

۱۹۸ for (۲-۶) حلقه در پایتون
۱۹۹`range` (۲-۶) تابع
۲۰۰for (۲-۲-۶) بخش else در حلقه
۲۰۲for (۳-۲-۶) مثال‌های مربوط به حلقه
۲۰۸(۳-۶) حلقه‌های تو در تو
۲۰۸(۴-۳-۶) شیوه‌ی کار حلقه‌های تو در تو
۲۰۹(۴-۳-۶) کاربردها و موارد استفاده حلقه‌های تو در تو
۲۱۱(۷) مقدمه‌ای بر توابع در زبان برنامه‌نویسی پایتون
۲۱۱(۷-۱) تعریف و هدف از توابع
۲۱۴(۷-۲) مزایای استفاده از توابع
۲۱۴(۷-۳) توابع داخلی و توابع تعریف‌شده توسط کاربر
۲۱۴(۷-۴) نحوه‌ی اجرا و فرآخوانی توابع
۲۱۵(۷-۴-۱) فرآخوانی تابع با پرانتز
۲۱۶(۷-۴-۲) فرآخوانی تابع بدون پرانتز
۲۱۷(۷-۵) پارامترها و آرگومان‌ها در توابع
۲۱۸(۷-۵-۱) تعریف پارامترها
۲۲۳(۷-۵-۲) تعریف آرگومان
۲۲۵(۷-۵-۳) مثال و تفاوت میان پارامترها و آرگومان‌ها

۲۲۶ آرگومان‌های دلخواه ۴-۵-۷
۲۳۲ ۷-۶) توابع درونی در زبان برنامه‌نویسی پایتون
۲۳۶ ۷-۷) آشنایی با <code>lambda</code> در زبان برنامه‌نویسی پایتون
۲۳۷ ۷-۷-۱) مثال‌های مربوط به <code>lambda</code>
۲۴۳ ۷-۸) آشنایی با مهم‌ترین توابع درونی <code>map()</code> , <code>filter()</code> , <code>zip()</code> در پایتون
۲۴۳ ۷-۸-۱) تابع درونی <code>zip()</code>
۲۴۴ ۷-۸-۲) تابع درونی <code>map()</code>
۲۴۷ ۷-۸-۳) تابع درونی <code>filter()</code>
۲۵۰ ۸) خطاهای و انواع آن‌ها در زبان برنامه‌نویسی پایتون
۲۵۰ ۸-۱) خطاهای نحوی
۲۵۱ ۸-۲) خطاهای زمان اجرا
۲۵۳ ۸-۳) خطاهای منطقی
۲۵۴ ۸-۴) سایر خطاهای متداول در پایتون
۲۶۰ ۹) مدیریت خطاهای در پایتون
۲۶۰ ۹-۱) ساختار <code>try/except</code> برای مدیریت کردن خطاهای
۲۶۱ ۹-۲) مدیریت چندین نوع خطا
۲۶۲ ۹-۳) استفاده از بلوک عمومی <code>except</code>
۲۶۳ ۹-۴) ترکیب با <code>else</code> و <code>finally</code>

۲۶۵	۵-۹) کاربرد مدیریت استثنایا.....
۲۶۷	۱) مفهوم دامنه در زبان برنامهنویسی پایتون.....
۲۶۷	۱۰) ا نوع Scope در پایتون
۲۶۷	۱۰-۱-۱) دامنه محلی
۲۶۸	۱۰-۱-۲) دامنه بسته
۲۶۹	۱۰-۱-۳) دامنه سراسری
۲۶۹	۱۰-۱-۴) دامنه داخلی.....
۲۶۹	۱۰-۲) قانون جستجوی نامها یا قانون LEGB
۲۷۰	۱۰-۳) متغیرهای سراسری و کلیدواژه global
۲۷۱	۱۰-۴) متغیرهای محلی و کلیدواژه nonlocal
۲۷۳	۱۱) برنامهنویسی شئگرا.....
۲۷۴	۱۱-۱) کلاس و شئ در زبان برنامهنویسی پایتون
۲۷۴	۱۱-۱-۱) کلاسها در پایتون.....
۲۷۵	۱۱-۱-۲) شئ در پایتون.....
۲۷۵	۱۱-۱-۳) ویژگیها و رفتارهای کلاس و اشیاء.....
۲۷۵	۱۱-۱-۴) مثالهایی از کلاس و شئ
۲۷۶	۱۱-۱-۵) مزایای استفاده از کلاس و شئ در پایتون
۲۷۶	۱۱-۱-۶) تعریف کلاس و شئ در پایتون

۲۷۷	تابع <code>__init__()</code>	۷-۱-۱۱
۲۸۲	تابع <code>__str__()</code>	۸-۱-۱۱
۲۸۳	تابع <code>__repr__()</code>	۹-۱-۱۱
۲۸۴	تفاوت بین <code>__str__()</code> و <code>__repr__()</code>	۱۰-۱-۱۱
۲۸۶	توابع وابسته در کلاس‌های پایتون	۱۱-۱-۱۱
۲۹۰	وراثت در پایتون	۱۱-۲-۱۱
۲۹۰	ویژگی‌های کلیدی وراثت	۱۱-۲-۱۱
۲۹۰	نحوه استفاده از وراثت در پایتون	۱۱-۲-۱۱
۲۹۱	مثالی از وراثت تک‌گانه	۱۱-۲-۱۱
۲۹۲	تابع وابسته‌ی <code>super()</code>	۱۱-۲-۱۱
۲۹۳	مثالی از وراثت چندگانه	۱۱-۲-۱۱
۲۹۴	مثالی از وراثت چندسطحی	۱۱-۲-۱۱
۲۹۵	چندريختی در پایتون	۱۱-۳-۱۱
۲۹۶	انواع چندريختی در پایتون	۱۱-۳-۱۱
۲۹۶	ویژگی‌های چندريختی	۱۱-۳-۱۱
۲۹۶	مثال‌های مربوط به مفهوم چندريختی	۱۱-۳-۱۱
۲۹۸	مزایای چندريختی	۱۱-۳-۱۱
۳۰۰	ماژول‌ها در پایتون	۱۲

۳۰۰	۱-۱۲) چرا باید از مازول‌ها استفاده کنیم؟
۳۰۰	۲-۱۲) انواع مازول‌ها
۳۰۰	۱-۲-۱۲) مازول‌های داخلی
۳۰۱	۲-۲-۱۲) مازول‌های تعریف شده توسط کاربر
۳۰۱	۳-۲-۱۲) مازول‌های شخص ثالث
۳۰۱	۳-۳-۱۲) وارد کردن مازول‌ها به داخل پروژه
۳۰۱	۱-۳-۱۲) وارد کردن پایه‌ای
۳۰۲	۲-۳-۱۲) وارد کردن مقدار خاص مانند کلاس از یک مازول
۳۰۲	۳-۳-۱۲) وارد کردن تمامی مقادیری که در داخل مازول وجود دارد
۳۰۲	۴-۳-۱۲) وارد کردن مازول با دادن نام مستعار برای آن مازول
۳۰۳	۵-۳-۱۲) مثال‌های مربوط به وارد کردن مازول‌ها
۳۰۳	۴-۱۲) ایجاد و استفاده از مازول
۳۰۴	۵-۱۲) متغیر <code>name</code> در پایتون
۳۰۴	۱-۵-۱۲) چگونگی عملکرد <code>name</code>
۳۰۴	۲-۵-۱۲) کاربرد <code>name</code>
۳۰۵	۳-۵-۱۲) مثال‌های مربوط به <code>name</code>
۳۰۷	۴-۵-۱۲) مزایای استفاده از <code>name</code>
۳۱۷	۱۴) آشنایی با مازول <code>Math</code> در پایتون

۳۱۷	۱-۱۴) ویژگی‌های مازول math
۳۱۸	۲-۱۴) کاربردهای مازول Math
۳۲۱	۳-۱۴) توابع وابسته موجود در مازول math
۳۲۲	۴-۱۴) توابع مهم و پرکاربرد مازول math
۳۲۲	۱-۴-۱۴) محاسبات عددی
۳۲۴	۲-۴-۱۴) توابع مثلثاتی.
۳۲۶	۳-۴-۱۴) لگاریتم‌ها و توابع نمایی
۳۲۸	۴-۴-۱۴) توابع ویژه
۳۳۰	۵-۴-۱۴) مدیریت اعداد اعشاری و صحیح
۳۳۲	۶-۴-۱۴) ثابت‌های ریاضی
۳۳۴	۵-۱۴) مثال‌هایی از کاربردهای عملی
۳۳۷	۱-۱۵) ویژگی‌های مازول datetime
۳۳۹	۲-۱۵) کاربردهای مازول datetime
۳۴۰	۳-۱۵) توابع وابستهٔ موجود در مازول Datetime
۳۴۲	۴-۱۵) توابع مهم و پرکاربرد مازول Datetime
۳۴۲	۱-۴-۱۵) برگرداندن تاریخ و زمان فعلی سیستم
۳۴۲	۲-۴-۱۵) برگرداندن تاریخ و زمان فعلی سیستم بدون تنظیمات منطقه زمانی
۳۴۳	۳-۴-۱۵) برگرداندن تاریخ و زمان فعلی در قالب UTC

۳۴۳.....	۴-۴) تبدیل یک رشته تاریخ/زمان به یک شیء <code>datetime</code>
۳۴۳.....	۴-۵) تبدیل یک شیء <code>datetime</code> را به رشته تاریخ/زمان
۳۴۴.....	۴-۶) برگرداندن تاریخ و زمان معادل یک زمان یونیکس (تعداد ثانیه‌ها از ۱ ژانویه ۱۹۷۰)
۳۴۴.....	۴-۷) ترکیب یک تاریخ و زمان و ایجاد یک شیء <code>datetime</code>
۳۴۵.....	۴-۸) تغییر دادن قسمتی از یک شیء <code>datetime</code>
۳۴۵.....	۴-۹) برگرداندن زمان یونیکس معادل یک شیء <code>datetime</code>
۳۴۸.....	۱۶) آشنایی با ماژول نامپای(NumPy) در پایتون
۳۴۸.....	۱۶-۱) ویژگی‌های اصلی نامپای
۳۴۹.....	۱۶-۲) کاربردهای نامپای
۳۴۹.....	۱۶-۳) مزایای نامپای
۳۴۹.....	۱۶-۴) مقایسه با فهرست‌ها در پایتون
۳۵۰.....	۱۶-۵) نصب نامپای
۳۵۰.....	۱۶-۱) نصب ماژول نامپای با pip
۳۵۰.....	۱۶-۲) بررسی نصب ماژول نامپای
۳۵۰.....	۱۶-۳) نصب نسخه خاص از ماژول نامپای
۳۵۱.....	۱۶-۴) بروزرسانی ماژول نامپای
۳۵۱.....	۱۶-۵) نصب ماژول نامپای در محیط مجازی
۳۵۱.....	۱۶-۶) ماتریس‌ها در نامپای

۳۵۱	۱۶-۶-۱) ویژگی‌های ماتریس‌ها در نامپایی.....
۳۵۲	۱۶-۶-۲) ایجاد ماتریس در ماژول نامپایی.....
۳۵۳	۱۶-۶-۳) عملیات‌های اصلی ماتریسی.....
۳۵۴	۱۶-۶-۴) معکوس ماتریس.....
۳۵۵	۱۶-۶-۵) دترمینان ماتریس.....
۳۵۶	۱۶-۶-۶) محاسبه مقادیر ویژه و بردارهای ویژه.....
۳۵۷	۱۶-۶-۷) دسترسی به عناصر ماتریس.....
۳۵۸	۱۶-۶-۸) تغییر شکل ماتریس:.....
۳۵۹	۱۶-۷-۱) مثال‌های کاربردی نامپایی.....
۳۶۰	۱۶-۷-۲) ایجاد آرایه.....
۳۶۱	۱۶-۷-۳) ایجاد آرایه‌های خاص.....
۳۶۲	۱۶-۷-۴) شکل و اندازه آرایه.....
۳۶۳	۱۶-۷-۵) عملیات عددی.....
۳۶۴	۱۶-۷-۶) توابع ریاضی.....
۳۶۵	۱۶-۷-۷) ایندکس و برش.....
۳۶۶	۱۶-۷-۸) تغییر شکل آرایه.....
۳۶۷	۱۶-۷-۹) عملیات ماتریسی.....
۳۶۸	۱۷) آشنایی با ماژول SymPy در پایتون.....

۳۶۳	۱-۱۷) ویژگی‌های SymPy
۳۶۴	۲-۱۷) کاربردهای SymPy
۳۶۵	۳-۱۷) نقاط قوت SymPy
۳۶۶	۴-۱۷) نصب و وارد کردن ماثول SymPy
۳۶۶	۵-۱۷) امکانات SymPy
۳۶۶	۱-۵-۱۷) ایجاد متغیرهای نمادین
۳۶۶	۲-۵-۱۷) سادهسازی و گسترش عبارات
۳۶۷	۳-۵-۱۷) مشتق و انتگرال
۳۶۷	۴-۵-۱۷) حل معادلات
۳۶۸	۵-۵-۱۷) رسم نمودار
۳۶۸	۶-۱۷) تفاوت NumPy با SymPy
۳۶۸	۷-۱۷) مثال‌های کاربردی SymPy
۳۶۸	۱-۷-۱۷) سادهسازی عبارات ریاضی
۳۶۹	۲-۷-۱۷) حل معادلات جبری (حل معادله درجه دوم)
۳۷۰	۳-۷-۱۷) محاسبه مشتق یکتابع
۳۷۰	۴-۷-۱۷) انتگرال‌گیری از یکتابع
۳۷۱	۵-۷-۱۷) حل معادلات دیفرانسیل
۳۷۱	۶-۷-۱۷) کار با ماتریس‌ها (محاسبه دترمینان یک ماتریس)

۳۷۲	۷-۷-۱۷) محاسبه مقادیر ویژه و بردارهای ویژه.....
۳۷۲	۸-۷-۱۷) سری تیلور.....
۳۷۳	۹-۷-۱۷) رسم نمودار یکتابع.....
۳۷۳	۱۰-۷-۱۷) کاربرد در هندسه (محاسبه مساحت دایره).....
۳۷۶	۱۸) آشنایی با ماژول pandas در پایتون.....
۳۷۶	۱-۱۸) ویژگی‌های اصلی Pandas.....
۳۷۸	۲-۱۸) کاربردهای Pandas.....
۳۷۸	۳-۱۸) مزایای استفاده از Pandas.....
۳۷۹	۴-۱۸) مقایسه Pandas با سایر کتابخانه‌ها.....
۳۷۹	۵-۱۸) نصب و وارد کردن ماژول Pandas به پروژه.....
۳۸۰	۶-۱۸) مثال‌های کاربردی ماژول Pandas در پایتون.....
۳۸۰	۶-۱۸) ۱-) بارگذاری داده‌ها از فایل CSV و نمایش داده‌ها.....
۳۸۰	۶-۱۸) ۲-) تمیز کردن داده‌ها (شناسایی و حذف داده‌های گمشده).....
۳۸۱	۶-۱۸) ۳-) مرتب‌سازی داده‌ها (مرتب‌سازی داده‌ها بر اساس یک ستون).....
۳۸۱	۶-۱۸) ۴-) گروه‌بندی داده‌ها (محاسبه مجموع داده‌ها در گروه‌های مختلف).....
۳۸۱	۶-۱۸) ۵-) محاسبات آماری (محاسبه میانگین، کمینه و بیشینه داده‌ها).....
۳۸۲	۶-۱۸) ۶-) انتخاب داده‌ها (فیلتر کردن داده‌ها بر اساس یک شرط).....
۳۸۲	۶-۱۸) ۷-) ایجاد ستون جدید (ایجاد یک ستون جدید بر اساس محاسبات ستون‌های موجود).....

۳۸۳.....	۸-۶) ترکیب و ادغام داده‌ها
۳۸۳.....	۹-۶) تغییر فرمت داده‌ها (تبديل تاریخ‌ها به فرمت زمانی)
۳۸۳.....	۱۰-۶) تحلیل داده‌های زمانی (محاسبه داده‌ها بر اساس بازه‌های زمانی)
۳۸۴.....	۱۱-۶) ذخیره داده‌ها (ذخیره داده‌های ویرایش شده به یک فایل جدید)
۳۸۴.....	۱۲-۶) مصورسازی ساده با Pandas (رسم نمودار از داده‌ها)
۳۸۵.....	۱۳-۶) محاسبه درصدها (محاسبه نسبت درصدی یک ستون)
۳۸۵.....	۱۴-۶) حذف ستون‌ها
۳۸۵.....	۱۵-۶) شناسایی و حذف داده‌های تکراری
۳۸۸.....	۱۹) آشنایی با ماژول Matplotlib در پایتون
۳۸۸.....	۱-۱۹) ویژگی‌های Matplotlib
۳۸۹.....	۲-۱۹) زیرماژول‌های Matplotlib
۳۸۹.....	۳-۱۹) کاربردهای Matplotlib
۳۹۰.....	۴-۱۹) ترسیم نمودارها با استفاده از ماژول Matplotlib در پایتون
۳۹۰.....	۱-۴-۱۹) ترسیم نمودار خطی (Line Plot)
۳۹۱.....	۲-۴-۱۹) ترسیم نمودار میله‌ای (Bar Plot)
۳۹۲.....	۳-۴-۱۹) ترسیم نمودار دایره‌ای (Pie Chart)
۳۹۲.....	۴-۴-۱۹) ترسیم نمودار پراکندگی (Scatter Plot)
۳۹۳.....	۴-۴-۱۹) ترسیم هیستوگرام (Histogram)

۳۹۴ ترسیم چند نمودار در یک صفحه ۶-۴-۱۹
۳۹۵ (۳D Plot) ترسیم نمودار سه بعدی ۷-۴-۱۹
۳۹۶ (۸) تنظیمات پیشرفته ۸-۴-۱۹
۳۹۶ (۵) محورها و برچسبها در مازول Matplotlib ۵-۱۹
۳۹۷ (۱) برچسب‌گذاری محورها ۱-۵-۱۹
۳۹۸ (۲) تنظیم محدوده محورها ۲-۵-۱۹
۳۹۸ (۳) اضافه کردن شبکه به محورها ۳-۵-۱۹
۳۹۹ (۴) برچسب‌های دلخواه برای محورها ۴-۵-۱۹
۴۰۰ (۵) چرخش برچسب‌های محورها ۵-۵-۱۹
۴۰۰ (۶) افزودن خطوط به محورها ۶-۵-۱۹
۴۰۱ (۷) اضافه کردن عناوین به محورها ۷-۵-۱۹
۴۰۲ (۸) تنظیمات گرافیکی پیشرفته محورها ۸-۵-۱۹
۴۰۲ (۶) ایجاد چندین نمودار در یک صفحه ۶-۱۹
۴۰۲ (۱) ساختار کلی subplot() ۱-۶-۱۹
۴۰۳ (۲) رسم چند نمودار ساده ۲-۶-۱۹
۴۰۴ (۳) چند نمودار در چند ردیف و ستون ۳-۶-۱۹
۴۰۵ (۴) استفاده از Subplot و Axes برای Figure ۴-۶-۱۹
۴۰۶ (۵) ترکیب نمودارها با ابعاد مختلف ۵-۶-۱۹

۴۰۷	۶-۶) اشتراک‌گذاری محورهای مشترک
۴۰۸	۶-۷) استفاده از تنظیمات پیشرفته
۴۰۹	۷-۱) نمودارهای هیستوگرام و میله‌ای در مازول Matplotlib
۴۱۰	۷-۲) نمودار هیستوگرام (Histogram)
۴۱۱	۷-۳) تفاوت‌های اصلی بین Bar Chart و Histogram
۴۱۲	۷-۴) ترکیب نمودارهای هیستوگرام و میله‌ای

فصل اول

آشنایی با برنامهنویسی

۱. آشنایی با برنامه‌نویسی^۱ و زبان‌های برنامه‌نویسی^۲

برنامه‌نویسی، 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 نوشتن 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 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 ساخت‌یافته 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 تا طراحی هوش مصنوعی، برنامه‌نویسی فرصتی برای کشف و خلق فراهم می‌کند که همچنان به پیشرفت و گسترش جهان دیجیتال منجر می‌شود.

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 انگلیسی به زبان بین‌المللی و مشترک میان افراد مختلف 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، می‌نویسند. 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 می‌شود که 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 مرتكب 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 داریم.

جالب است بدانید که امروزه تمام سیستم‌هایی که کوچکترین اثری از هوشمندسازی در آن‌ها دیده می‌شود، برنامه‌نویسی شده‌اند. تک تک کارهایی که قادر هستید با استفاده از موبایل‌تان انجام دهید، مانند فرستادن پیام، تماس، پخش ویدیو، پخش موسیقی و... برنامه‌نویسی شده‌اند. امروزه اشیا و وسایل خانه نیز قابلیت برنامه‌نویسی دارند و نسل جدید لباسشویی، یخچال و... همگی با استفاده از این حرفه، هوشمندسازی شده‌اند.

۱-۵) تاریخچه برنامه‌نویسی

برنامه‌نویسی اولین بار در سال ۱۸۸۳ توسط یک ریاضی‌دان و نویسنده انگلیسی به نام ایدا لاولیس، زمانی که بر روی پروژه موتور تحلیلی دانشمند معروف چارلز ببیج 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 رخ داد تا مقدمه ساز ایجاد زبان‌های برنامه‌نویسی شود. در دهه ۱۹۵۰ زبان اسembly^۳ 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^۴ که با هدف حل محاسبات علمی و ساده‌سازی عملیات ریاضی طراحی شده بود، ظهرور کرد و به عنوان اولین زبان برنامه‌نویسی علمی شناخته شد. از سال ۱۹۷۰ زبان‌های برنامه‌نویسی مدرن مانند پایتون، جاوا اسکریپت، روئی و ... ساخته شدند و تا امروز که زبان‌های متعدد دیگری به وجود آمده‌اند.

سخنانی از افراد مشهور درباره برنامه‌نویسی

"هر فردی قادر به نوشتن کد برای ماشین‌آلات کامپیوتری و دیجیتالی هست، اما تفاوت در زمانی است که برنامه‌نویسان خبره بتوانند کد تولیدی خود برای افراد تشریح و قابل فهم نمایند." مارتین فولر

"در آینده‌ای نزدیک، برنامه‌نویسی حکم آموزش‌های اولیه هم‌چون نوشتن و خواندن را برای کودکان پیدا می‌کند و نیاز به آن در همه زمینه‌های احساس خواهد شد، به‌گونه‌ای که بزرگ‌ترها افسوس می‌خورند چرا زودتر اقدام به آموزش آن نکرده‌اند." مارک زاکربرگ

³ Assembly Language

⁴ Fortran

"سنجهش میزان برتری در کدنویسی با تعداد خطوط کد مثل سنجش میزان وزن در حین ساخت هواپیماست؛

پس هرچقدر کمتر باشد بهتر است!" بیل گیتس

۱-۶) زبان‌های برنامه‌نویسی

دنیای زبان‌های برنامه‌نویسی: دروازه‌ای به سوی خلق ایده‌ها؛

برنامه‌نویسی هنر ترجمه ایده‌ها به زبان کامپیوتر است. با استفاده از زبان‌های برنامه‌نویسی، دستوراتی به کامپیوتر می‌دهیم تا وظایف مختلفی را انجام دهد، مانند راهاندازی یک وبسایت ساده یا طراحی 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 (انگلیسی) است. بر همین اساس هر زبان برنامه‌نویسی در یکی از دو گروه سطح بالا و سطح پایین قرار می‌گیرد.

زبان‌های برنامه‌نویسی سطح پایین^۵

زبان‌های سطح پایین، 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

⁵ Low-level programming language

زبان‌های سطح پایین می‌توان به زبان‌هایی مانند اس‌مبلی، C و Cpp اشاره کرد و اما در مقابل زبان‌های برنامه‌نویسی

سطح پایین،

زبان‌های برنامه‌نویسی سطح بالا⁶

زبان‌های سطح بالا با زبانی نزدیک به زبان انسان، دستورات را به کامپیوتر ابلاغ می‌کنند. گویی با این زبان‌ها، از پیچیدگی‌های ریز و جزئیات فنی فاصله می‌گیریم. 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، می‌توانیم بر منطق برنامه و کارکرد اصلی آن تمرکز کنیم و درگیر پیچیدگی‌های فنی سطح پایین نشویم. 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.

✓ سی‌پلاس‌پلاس: زبان سریع و کارآمد برای برنامه‌های سیستمی، بازی‌ها و شبیه‌سازی‌ها.

✓ سی‌شارپ: زبان قدرتمند برای برنامه‌نویسی تحت ویندوز و وب‌اپلیکیشن‌ها.

✓ پی‌اچ‌پی: زبان محبوب برای برنامه‌نویسی سمت سرور و وب‌سایت‌های پویا.

✓ متلب: زبان قدرتمند برای استفاده در کاربردهای مهندسی و محاسبات پیچیده ریاضی.

⁶ High-level programming language

۱-۶-۱) آشنایی با زبان برنامه‌نویسی پایتون^۷

زبان برنامه‌نویسی پایتون در سال ۱۹۹۱ توسعه شخصی به نام خیدو فان روسم^۸ توسعه داده شد. پایتون یک زبان برنامه‌نویسی شی گرا^۹ و سطح بالا است؛ سطح بالا بودن پایتون یکی از مهمترین دلایل محبوبیت آن 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 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 کامپیوتری استفاده کرد.

۱-۶-۲) آشنایی با زبان برنامه‌نویسی سی^{۱۰}

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 می‌باشد. زبان سی یکی از پرکاربردترین زبان‌های برنامه‌نویسی در دنیا می‌باشد که از اهمیت بالایی در علوم کامپیوتر برخوردار است چراکه زبان سی مادر همه زبان‌های برنامه‌نویسی می‌باشد و از طرفی ریشه و اساس زبان‌هایی مثل سی پلاس پلاس، C، Objective-C، C#، PHP، پایتون و ... به حساب می‌آید. با اینکه حدود ۵۰ سال از زمان به وجود آمدن این زبان می‌گذرد اما همچنان زبان برنامه‌نویسی سی در بین ۵ تا ۱۰ زبان محبوب و پرطرفدار در دنیا قرار دارد. دنیس ریچی در بین سال‌های ۱۹۶۹ تا ۱۹۷۳، زبان برنامه‌نویسی سی را ساخت.

⁷ Python Programming Language

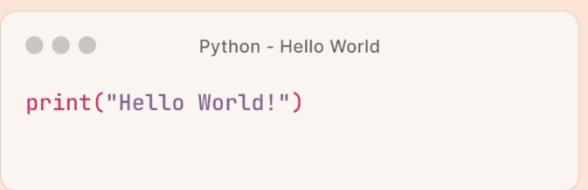
⁸ Guido van Rossum

⁹ Object Oriented Programming

¹⁰ C Programming Language

زبان برنامه‌نویسی سی انتخابی مناسب برای برنامه‌نویسی سیستمی محسوب می‌شود. منظور از برنامه‌نویسی سیستمی توسعه سیستم‌های عامل، کامپایلرها و درایورهای شبکه است. این زبان با اینکه از محبوبیت زیادی برخوردار است ولی در عین حال برخی از افراد عقیده دارند که ساختار زبان سی برای یادگیری پیچیده است.

تفاوت زبان برنامه‌نویسی پایتون با سی

C	پایتون
زبان برنامه‌نویسی سی توسعه دنیس ریچی در سال ۱۹۷۲ توسعه یافت.	زبان برنامه‌نویسی پایتون اولین بار توسط خیدو فان روسم در سال ۱۹۹۱ توسعه یافت.
اجرای برنامه‌های نوشته شده با سی سریع‌تر هستند.	اجرای برنامه‌های نوشته شده با پایتون کندتر هستند.
مدیریت حافظه باید به صورت دستی در سی انجام شود. (می‌توان حافظه را مدیریت کرد).	مدیریت حافظه در پایتون بصورت خودکار انجام می‌شود. (نمی‌توان حافظه را مدیریت کرد).
زبان برنامه‌نویسی سی بیشتر برای توسعه برنامه‌های سخت‌افزاری استفاده می‌شود.	پایتون یک زبان برنامه‌نویسی عمومی است (می‌توان برای کارهای مختلف از پایتون استفاده کرد).
کدهای سی با پسوند .C. ذخیره می‌شوند.	کدهای پایتون با پسوند .py. ذخیره می‌شوند
یادگیری نسبتاً دشوار	یادگیری آسان
برنامه چاپ Hello World! در سی	برنامه چاپ Hello World! در پایتون
 <pre> ... C - Hello World #include <stdio.h> int main() { printf("Hello World!"); return 0; } </pre>	 <pre> ... Python - Hello World print("Hello World!") </pre>

۱-۶-۳) آشنایی با زبان برنامه‌نویسی سی پلاس پلاس^{۱۱}

زبان سی پلاس یک زبان برنامه‌نویسی کامپیوتری می‌باشد که 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 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 سی پلاس پلاس 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.

تفاوت زبان برنامه‌نویسی پایتون با سی پلاس پلاس

C++	پایتون
سی پلاس پلاس معمولاً دارای خطوط طولانی کد است.	پایتون خطوط کد کمتری دارد.
ساختار کد نسبتاً دشواری دارد.	ساختار کد آسانی دارد.
سرعت بالایی در اجرای کد دارد.	سرعت کمتری در اجرای کد دارد.
از کاربردهای آن می‌توان به توسعه بازی، طراحی سیستم عامل و ... اشاره کرد.	از کاربردهای آن می‌توان به توسعه وب، تجزیه و تحلیل داده‌ها، محاسبات علمی و ... اشاره کرد.
کدهای سی پلاس پلاس با پسوند .cpp. ذخیره می‌شوند.	کدهای پایتون با پسوند .py. ذخیره می‌شوند.
یادگیری نسبتاً دشوار	یادگیری آسان
برنامه چاپ Hello World! در سی پلاس پلاس	برنامه چاپ Hello World! در پایتون

^{۱۱} Cpp(C++) Programming Language

```

C++ - Hello World
#include <iostream>
int main() {
    std::cout << "Hello World!";
    return 0;
}

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

```

۱-۶-۴) آشنایی با زبان برنامه‌نویسی سی شارپ^{۱۲}

بدون شک یکی از محبوب‌ترین و پرکاربردترین زبان‌های برنامه‌نویسی حال حاضر دنیا سی شارپ نام دارد و بر اساس آخرین تحقیقات صورت گرفته این زبان جزو ۵ زبان برنامه‌نویسی برتر در دنیا می‌باشد، که همچنین بازار کار بسیار خوبی در ایران دارد. برنامه‌نویسی به زبان سی شارپ، یک زبان برنامه‌نویسی مدرن است که از ویژگی شیء‌گرا بودن برخوردار بوده و توسط شرکت مایکروسافت در سال ۲۰۰۰ طراحی شده است. ریشه این زبان همان زبان سی است، اما برای برنامه‌نویسان جاوا و جاوا اسکریپت آشنایی است. به کمک سی شارپ، یک برنامه‌نویس به راحتی می‌تواند برنامه‌های دسکتاپ مبتنی بر ویندوز را ایجاد کند. نکته‌ی قابل توجه در مورد زبان سی شارپ که باعث برتری این زبان نسبت به سایر رقبا است، پشتونه‌ای بزرگ به نام مایکروسافت است. بر اساس گزارشی که در سال ۲۰۰۲ منتشر شد، مشخص شد که شرکت مایکروسافت پس از صرف دو میلیون دلار هزینه و ۵ میلیون ساعت کار بی وقفه توانسته این زبان برنامه‌نویسی را در اختیار توسعه دهندگان سراسر دنیا قرار دهد؛ به همین دلیل به جرات می‌توان گفت زبان برنامه‌نویسی سی شارپ آینده خوب و مطمئنی خواهد داشت.

تفاوت زبان برنامه‌نویسی پایتون با سی شارپ

C#

پایتون

^{۱۲} Csharp(C#) Programming Language

سی شارپ توسط مایکروسافت توسعه یافته است و حتی برای اهداف تجاری به صورت رایگان در دسترس است.	پایتون همچنین یک پلتفرم منبع باز ^{۱۳} است و حتی برای اهداف تجاری به صورت رایگان در دسترس است.
ساختار کد نسبتاً دشواری دارد.	ساختار کد آسانی دارد.
دارای خطوط طولانی کد است	خطوط کد کمتری دارد
از کاربردهای آن می‌توان به توسعه بازی، توسعه برنامه‌های دسکتاپ و ... اشاره کرد.	از کاربردهای آن می‌توان به توسعه وب، تجزیه و تحلیل داده‌ها، هوش مصنوعی و ... اشاره کرد.
کد های سی شارپ با پسوند .CS. ذخیره می‌شوند.	کدهای پایتون با پسوند .py. ذخیره می‌شوند
برنامه چاپ Hello World! در سی شارپ	برنامه چاپ Hello World! در پایتون

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

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

۱-۶-۵) آشنایی با زبان برنامه‌نویسی جاوا^{۱۴}

زبان برنامه‌نویسی جاوا در اوایل دهه ۱۹۹۰ میلادی توسط جیمز گاسلینگ در شرکت سان‌مایکروسیستمز پایه‌ریزی شد. در حقیقت، نارضایتی گاسلینگ از اصول برنامه‌نویسی در زبان سی‌پلاس‌پلاس و نارساپی‌های این زبان موجب

¹³ Open-Source

¹⁴ Java Programming Language

شد تا وی جawa 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 جawa 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.

تفاوت زبان برنامه‌نویسی جawa با پایتون

جاوا	پایتون
به طور کلی سریع‌تر از پایتون است، به خصوص برای برنامه‌های کاربردی بزرگ و پیچیده	کندتر از جawa، اما برای برخی از وظایف مانند اسکریپت‌نویسی و یادگیری ماشین به اندازه کافی سریع است
کمی دشوارتر از پایتون برای یادگیری	یادگیری آسان
مناسب برای توسعه برنامه‌های سازمانی، برنامه‌های دسکتاب، سیستم‌های تعابیه شده، و وب‌اپلیکیشن‌های سازمانی	مناسب برای توسعه وب، علم داده، یادگیری ماشین، هوش مصنوعی، و اسکریپت‌نویسی

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!")
```

۶-۶) آشنایی با زبان برنامه‌نویسی متلب^{۱۵}

زبان برنامه‌نویسی متلب 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 هنگامی 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، بهمراه همکارانش، شرکت MathWorks را تأسیس کردند. این شرکت اولین نسخه‌ی رسمی خود از متلب را در سال ۱۹۸۴ منتشر کرد.

متلب یک زبان نسل‌چهارم (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. می‌توانیم داده‌ها را از منابع مختلف، مانند فایل‌ها، پایگاه‌داده‌ها یا وب، آپلود کنیم تا آن‌ها را تجزیه و تحلیل کنیم و بعد با استفاده از طیف وسیعی از نمودارهای گرافیکی که در متلب وجود دارد آن‌ها را نمایش دهیم؛ علاوه بر این، متلب کتابخانه‌ای از توابع ریاضی را شامل می‌شود که به ما امکان می‌دهد عملیاتی مانند جبر خطی و محاسبات ماتریس‌ها را انجام دهیم.

^{۱۵} Matlab Programming Language

تفاوت زبان برنامه‌نویسی متلب با پایتون

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 عمده‌اً 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 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 گسترده 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 لیست‌ها، دیکشنری‌ها و مجموعه‌ها، 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 بر جسته پایتون تایپ پویا و اجرای تفسیری آن است. این خصوصیات امکان آزمایش و توسعه سریع برنامه‌ها را فراهم می‌کنند 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 پژوهه‌های علمی و تحقیقاتی، پایتون به دلیل سهولت استفاده و ابزارهای گسترده، 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 و سینتکس 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 بگیرد. از همان سالی که خیدو با گروه ABC کار می‌کرد در فکر تحقق بخشیدن به پروژه تیم ABC بود؛ اینکه بتواند زبانی ساده، با طراحی بهتر از زبان قدرتمندی نظری C پیاده‌سازی کند. ترکیب C و انگیزه توسعه و رشد سیستم‌عامل Amoeba، سبب شد تا خیدو فان روسوم به دنبال تحقق 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 تبدیل کنند. در جدول ۱، به بررسی مهم‌ترین ویژگی‌های زبان برنامه‌نویسی پایتون می‌پردازیم.

ویژگی	توضیحات
سادگی و خوانایی بالا	یکی از ویژگی‌های برجسته پایتون، سادگی در نوشتار و خوانایی کد است. از این زبان می‌توان به عنوان "زبانی که با انسان‌ها صحبت می‌کند" 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 ^{۱۶} یاد می‌کنند.
جامعه فعال و گسترده	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.
منبع‌باز و رایگان	پایتون یک زبان منبع‌باز 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.

¹⁶ User Friendly

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 شارپ قرار دهید.

سازگاری با زبان‌های
دیگر

پایتون، مازول‌ها 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 باعث افزایش سرعت کدنویسی و صرفه‌جویی در زمان می‌شود.

مازول‌ها و
کتابخانه‌های فراوان

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.

سطح بالا بودن

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، برروی انواع سیستم‌عامل‌ها اجرا کرد.

پورتابل و قابل حمل
بودن

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.

زبان برنامه‌نویسی
تفسیر

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

شی‌گرایی

می‌توانید از پایتون در هر پروژه کوچک و بزرگ استفاده کنید (از نوشتن یک خط کد ساده گرفته تا نوشتن الگوریتم‌های هوش مصنوعی).

کاربردهای متنوع

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 ویژگی‌های این زبان قدرتمند آشنا شدیم. 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 همه‌کاره یاد می‌شود. از پایتون می‌توان برای ساخت پروژه‌های کوچک و بزرگ استفاده کرد که همین ویژگی‌ها و کاربردها، دلیلی برای محبوبیت پایتون شده است. در جدول ۲، به بررسی کاربردهای زبان برنامه‌نویسی پایتون می‌پردازیم.

کاربردها	توضیحات
هوش مصنوعی و یادگیری ماشین	امروزه هوش مصنوعی و یادگیری ماشین جزء مباحث روز دنیا می‌باشد. پایتون با کتابخانه‌ها و ابزارهای داخلی خود توسعه الگوریتم‌های هوش مصنوعی را آسان کرده است. علاوه بر این، کدهای ساده، مختصر و قابل خواندن را ارائه می‌دهد که نوشتن الگوریتم‌های پیچیده برای توسعه‌دهندگان را آسان‌تر می‌کند. از ابزارهای پایتون که برای هوش مصنوعی استفاده می‌شود، می‌توان به پایتورچ، کراس، تنسورفلو و... اشاره کرد.
توسعه برنامه‌های تحت وب	توسعه وب یکی از مهم‌ترین کاربردهای پایتون است؛ پایتون طیف گسترده‌ای از فریمورک‌ها و کتابخانه‌ایی نظری جنگو، فلسک و موارد دیگر ارائه می‌دهد که امکان سهولت را برای توسعه‌دهندگان فراهم می‌کند. معمولاً از پایتون برای برنامه‌نویسی سمت سرور استفاده می‌شود.

<p>برنامه‌های محبوب مانند گوگل، نتفلیکس و ردیت همگی از پایتون استفاده می‌کنند. پایتون اغلب به عنوان یک زبان پشتیبانی برای توسعه دهنده‌گان نرم افزار، برای کنترل و مدیریت ساخت و تست استفاده می‌شود.</p>	<p>توسعه نرم افزار</p>
<p>پایتون توانایی تجزیه و تحلیل و تجسم داده‌ها در قالب نمودارها را دارد و از ابزارهایی نظیر پانداس، نامپای، مت‌پلات‌لیب می‌توان این کارها را انجام داد.</p>	<p>تجزیه و تحلیل داده‌ها</p>
<p>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. البته می‌توان با ابزارهایی نظیر پانداس 3d برای ساخت بازی‌های سه‌بعدی استفاده کرد.</p>	<p>توسعه بازی</p>
<p>پایتون به توسعه‌دهنده‌گان کمک می‌کند تا رابط کاربری گرافیکی^{۱۷} را به راحتی توسط کتابخانه‌ها و فریمورک‌هایی نظیر پای‌کیوودی، کیوی و ... ساخته شود.</p>	<p>برنامه‌های دسکتاپ و موبایل</p>
<p>وب اسکرپینگ یا استخراج داده یک فرآیند خودکار است که برای استخراج اطلاعات از وبگاه‌ها به روشهای ساده و سریع انجام می‌گیرد. از ابزارهای پایتون نظیر اسکرپی و سلنیوم برای استخراج اطلاعات از وبگاه‌ها استفاده می‌شود.</p>	<p>استخراج داده از اینترنت</p>
<p>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.</p>	<p>پردازش تصویر</p>

¹⁷ Graphical User Interface (GUI)

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، باگ‌بانتی و ... استفاده کرد.

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 منبع باز نیست، اما این شرکت یک API بسیار قدرتمند با کد پایتون را راه اندازی کرده است. هم‌چنین توسعه دهنده‌گان دراپ باکس از پایتون در بیشتر برنامه‌نویسی‌های سمت سرور خود استفاده می‌کنند.

۵-۲) نصب و راه اندازی پایتون

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> دانلود کنید.

The screenshot shows the Python.org homepage with the navigation bar: Python, PSF, Docs, PyPI, Jobs, Community. Below the header is the Python logo and a search bar. The main content area displays a code snippet in the Python interpreter:

```
# Python 3: List comprehensions
>>> fruits = ['Banana', 'Apple', 'Lime']
>>> loud_fruits = [fruit.upper() for fruit in
fruits]
>>> print(loud_fruits)
['BANANA', 'APPLE', 'LIME']

# List and the enumerate function
>>> list(enumerate(fruits))
[(0, 'Banana'), (1, 'Apple'), (2, 'Lime')]
```

To the right, there's a section titled "Compound Data Types" with a brief description and a link to "More about lists in Python 3". Below the code, there are five numbered buttons (1, 2, 3, 4, 5). At the bottom, a banner states: "Python is a programming language that lets you work quickly and integrate systems more effectively. [» Learn More](#)".

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.



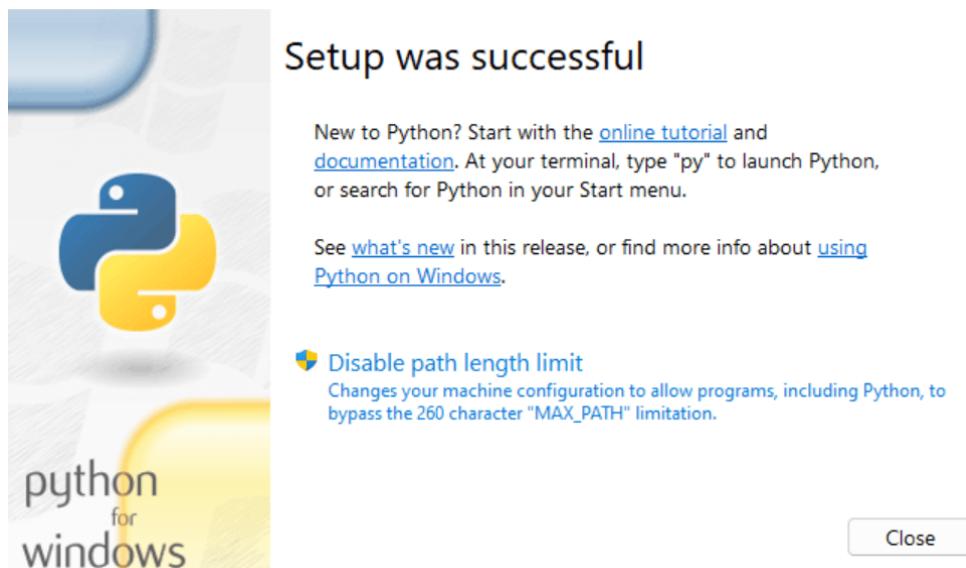
فرایند نصب پایتون

برای نصب پایتون، گزینه Install Now را انتخاب کنید(شکل بالا).

نکته: لازم است تیک گزینه Add python.exe to PATH 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، صفحه زیر ظاهر می‌شود.



اتمام نصب پایتون

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

۱. ترمینال یا CMD را در ویندوز خودتان باز کنید.
۲. سپس با زدن دستور `python -v` اطمینان حاصل کنید که پایتون نصب شده است و پیغام

زیر به شما نمایش داده می‌شود (شکل ۴).

```
Command Prompt
Microsoft Windows [Version 10.0.19045.4291]
(c) Microsoft Corporation. All rights reserved.

C:\Users\RezaS>python --version
Python 3.12.3

C:\Users\RezaS>
```

بررسی این که پایتون نصب شده است یا خیر.

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

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

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/> می‌روید.
- 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 پردازنده‌های Intel و دیگری برای پردازنده‌های Apple M1 و باید نسخه نصبی موردنظر را دانلود کرده و نصب کنید.
- فایل نصبی دانلود شده را اجرا کنید و طبق دستورالعمل‌هایی 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 می‌شود.

۲. نصب با Homebrew

Homebrew یک مدیر بسته محبوب برای سیستم عامل مک است که می‌تواند برای نصب آسان بسیاری از برنامه‌ها، از جمله پایتون، استفاده شود. اگر Homebrew را نصب ندارید، می‌توانید آن را با دستور زیر نصب کنید:

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

پس از نصب Homebrew، می‌توانید پایتون را با دستور زیر نصب کنید. این دستور آخرین نسخه پایتون را نصب می‌کند:

```
brew install python
```

همچنین می‌توانید نسخه خاصی از پایتون را با استفاده از دستور زیر نصب کنید (برای مثال نسخه ۳.۱.۰):

```
brew install python@3.10
```

نکته: اگر قبلاً 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:

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 VS Code دارای افزونه‌ها و ویژگی‌های بسیاری است که می‌تواند به

شما در کدنویسی پایتون کمک کند.

❖ Sublime Text: 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.

۲. محیط‌های توسعه یکپارچه^{۱۸}: 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

عبارةتند از:

❖ PyCharm^{۱۹}: 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^{۲۰}: 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.

¹⁸ Integrated Development Environment (IDE)

¹⁹ PyCharm

²⁰ Spyder

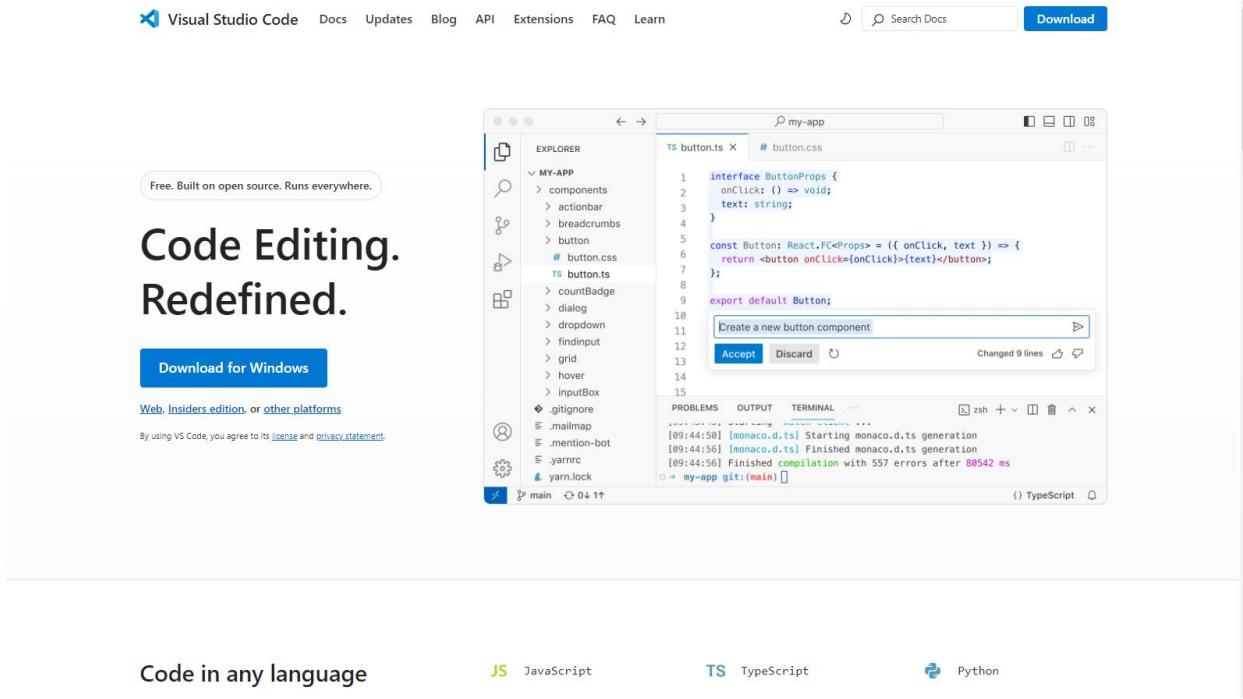
چند نکته اضافی برای استفاده از ویرایشگر کد برای اجرای کدهای پایتون:

- هنگامی که فایل پایتون خود را ذخیره می‌کنید، باید از پسوند `.py` استفاده کنید. به عنوان مثال، اگر فایل کدهای شما `app` نام دارد، باید آن را به صورت `app.py` ذخیره کنید.
- بعد از این‌که کد خود را نوشتید و فایل پایتون 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 `python3` در ترمینال استفاده کنید: به عنوان مثال، برای اجرای فایل `app.py`، از دستور زیر در CMD استفاده کنید.

```
python3 app.py
```

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



Code in any language

JS JavaScript

TS TypeScript

Python

ویگاه رسمی VS Code

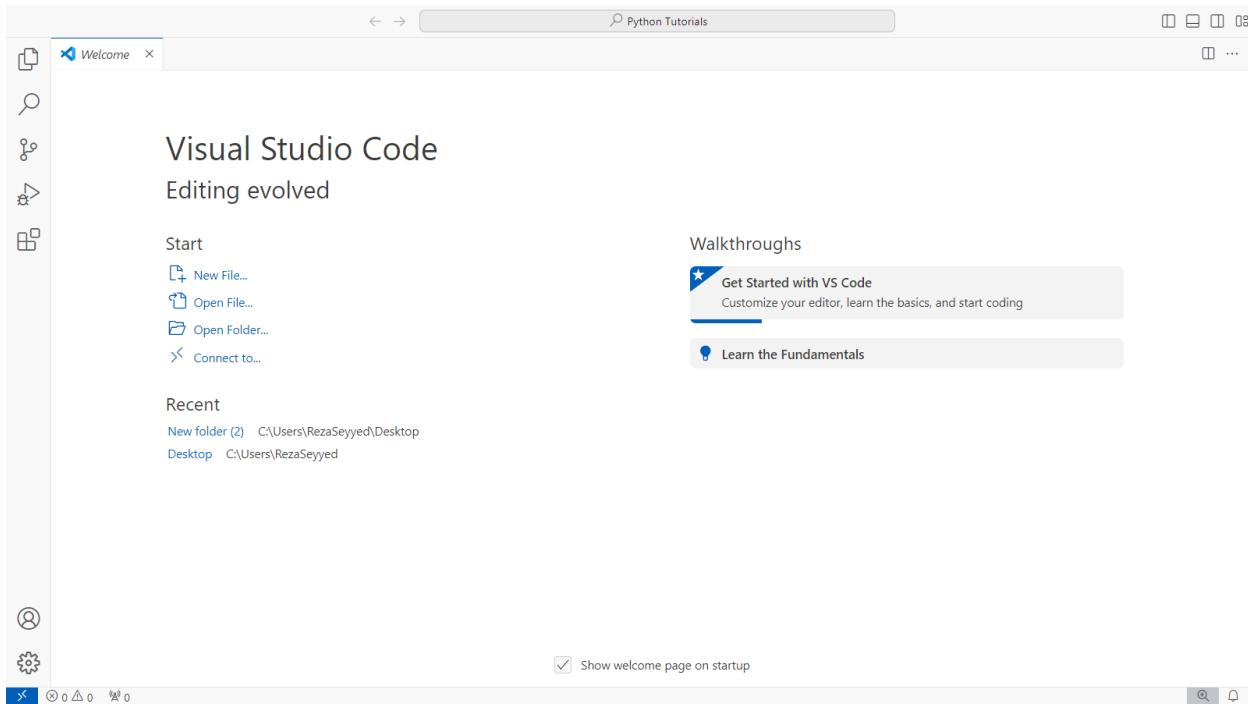
نکته: فرایند نصب آسان و همانند نصب پایتون می‌باشد. هنگام نصب VS Code تیک گزینه Add to PATH را

فعال کنید.

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

هنگامی که نرم‌افزار VS Code را باز می‌کنید با محیطی روبرو می‌شوید که می‌توانید کدهای خود را در آنجا

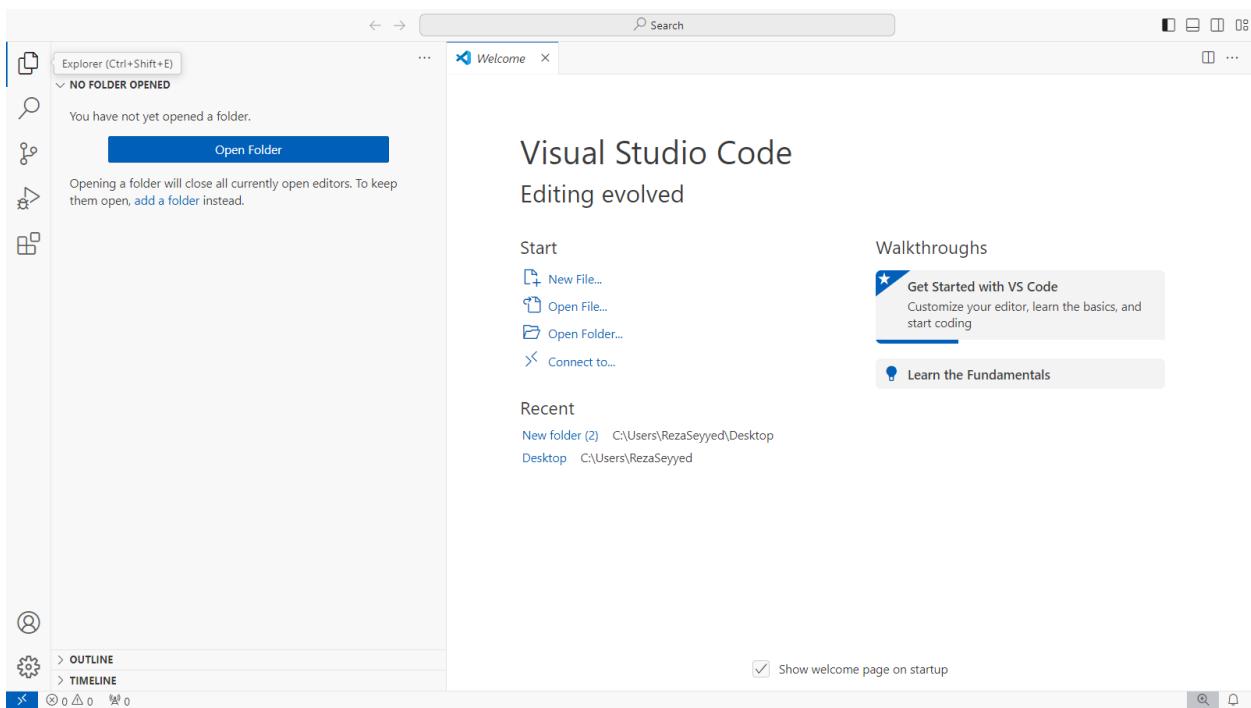
نوشته، ذخیره و اجرا کنید.



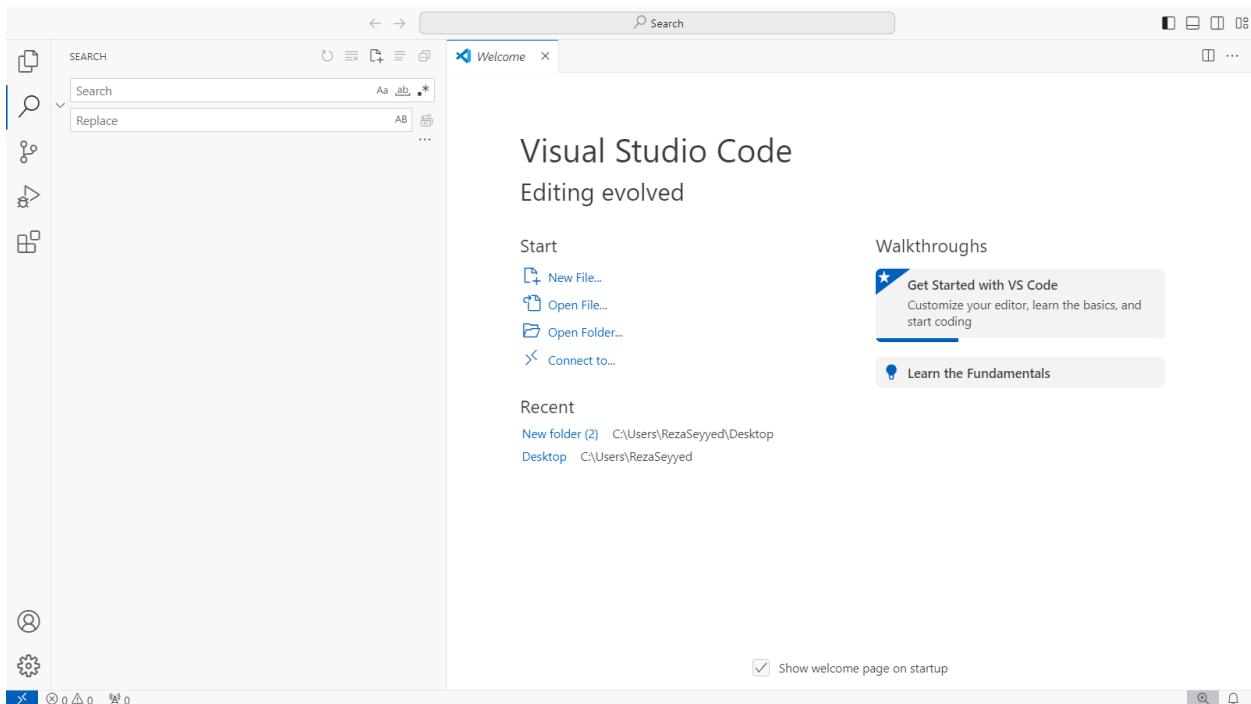
عکس از محیط VS Code

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

آیکون اول مربوط به قسمت Explorer می‌باشد که در این قسمت، فolderها و فایل‌ها قرار می‌گیرند(شکل پایین).



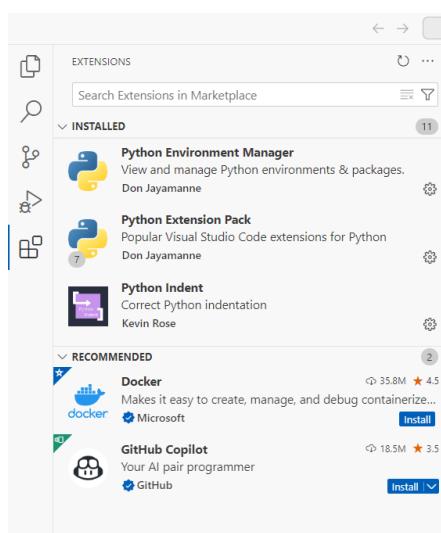
آیکون دوم مربوط به جستجوی فایل مورد نظر می‌باشد. اگر تعداد فایل‌هایی که در explorer ایجاد کردید زیاد باشد، از قسمت Search می‌توانید، فایل مورد نظرتان را جستجو کنید(شکل پایین).



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 مربوط به افزونه‌ها می‌باشد. معمولاً برای کدنی آسان و سریع و همچنین برخی تغییرات محیط VS Code از این قسمت استفاده می‌کنند و افزونه‌های مورد نظر را نصب می‌کنند.

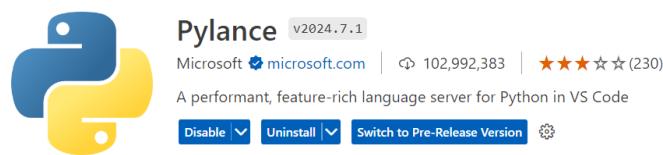


نکته: قبل از اجرای برنامه‌های پایتون، از قسمت افزونه‌ها در نرم‌افزار VS Code، چند افزونه را در نصب کنید. این افزونه‌ها برای کدنی سریع و آسان و همچنین اجرای کدهای پایتون نیاز و ضروری است.

:Python •



:Pylance •



• Python Indent

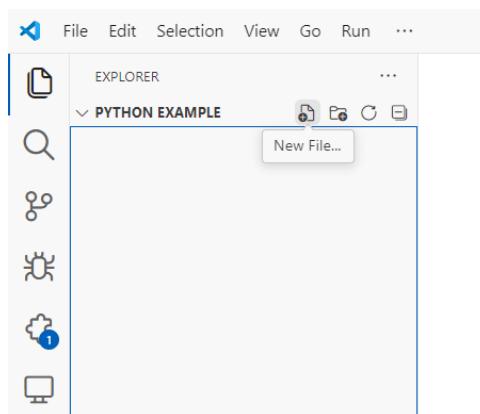


۷-۲) اجرای کدهای پایتون با VS Code

برای این که بتوانید کدهای پایتون را در یک فایلی بنویسید و آن فایل را در یک فolderی ذخیره کنید، ابتدا باید در صفحه اصلی دسکتاپ یک folder ایجاد کرده و نام‌گذاری کنید(نام folder باید انگلیسی باشد). سپس روی آن folder راست کلیک کرده و در نواربار ظاهر شده گزینه «Open With Code»(آیکون باز شده باشد) انتخاب کرده و منتظر باز شده VS Code بمانید.

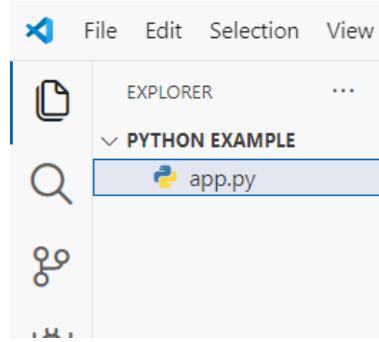
سپس از نوار سمت چپ VS Code، روی دکمه «New File» کلیک کنید تا یک فایل جدید ایجاد شود(شکل

پایین).



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

نکته: به طور خودکار فایل ایجاد شده را باز می‌کند.



بعد از اینکه فایل پایتون 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 انجام

این کار باید از دستور print طبق شکل ۱۳ استفاده کنید.

A screenshot of the Visual Studio Code code editor. The title bar shows 'File Edit Selection View Go Run ...'. The editor window displays a single line of Python code: 'print("Hello World!")'. The code is highlighted in blue, and the string 'Hello World!' is highlighted in red.

توضیحات شکل ۱۳:

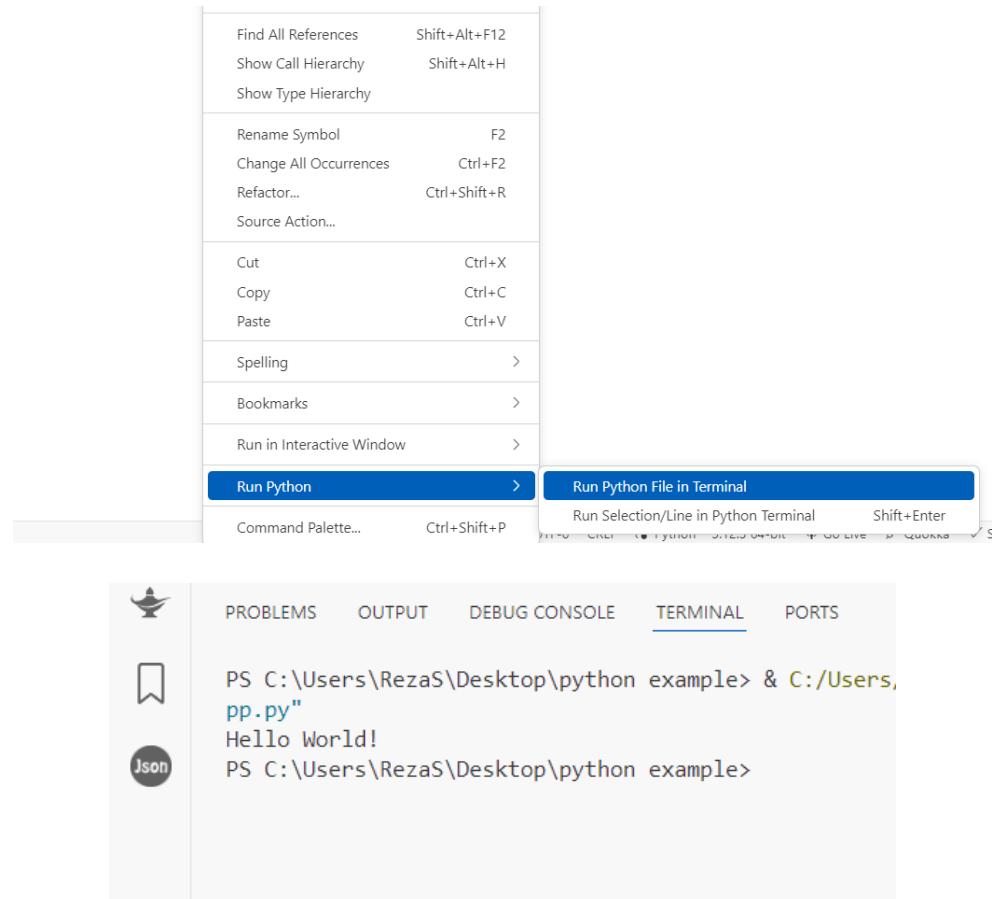
• دستور print برای چاپ متن در کنسول استفاده می‌شود.

• 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 روی گزینه Terminal کلیک کنید.



خروجی

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.

۱-۳) نحوه کامنت‌گذاری^{۲۱} 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. روش‌دیگری برای افزایش خوانایی کدها، نوشتن کامنت یا کامنت‌گذاری 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 تک خطی^{۲۲}
- کامنت‌گذاری چندخطی^{۲۳}

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.

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

²¹ Comment

²² Single-Line Comment

²³ Multiple Line Comment

```
1 # Comment in Python
2
3 # ** Single Line Comment **
4 # print("Hello World") کامنت تک خطی
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 """
```

نحوه کامنت‌گذاری تک خطی و چند خطی در پایتون

مزایای کامنت‌گذاری:

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

(۲-۳) آشنایی تابع چاپ (print)

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

این تابع پارامترهای مختلفی دارد که هر کدام مقدار (آرگومان) مشخصی می‌گیرند. مهم‌ترین پارامترها در **جدول** آمده است.

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

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=" ") # ()
```

مثالی از پارامتر sep در print

خروجی:

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: #
```

مثالی از پارامتر end در print

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

۳-۳ آشنایی با **f-string** در پایتون

پایتون یک ویژگی قدرتمند به نام رشته های f (رشته های قالب بندی شده) را برای ساده سازی قالب بندی و درون یابی رشته ها ارائه می دهد. Formatted string literals یا f-string روشی جدید برای قالب بندی رشته ها در پایتون می باشد. در پایتون f-string روشی برای نمایش دادن ترکیبی رشته ها و متغیرها است.

:f-string مزایای

- خوانایی و نوشتگی کد را آسان تر می کنند.
 - از قابلیت های قدرتمندی مانند جایگذاری متغیر، عبارات و حتی قالب بندی اعداد به صورت دلخواه پشتیبانی می کنند.
 - با انواع مختلف داده ها، از جمله رشته ها، اعداد، لیست ها و دیکشنری ها می توان کار کرد.
 - کوتاهی : f-string ها معمولاً کد را کوتاه تر و فشرده تر می کنند.
 - کاهش تعداد خطاهای اشتباهی را کاهش می دهند.
- در ادامه به مثالی از f-string توجه کنید.

python example - Introduction.py

```
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.")
```

مثالی از f-string در پایتون

خروجی:

My name is Reza and I am 23 years old.

I got a grade of 18.23 in math.

۴-۳) انواع عملگرها^{۲۴} در پایتون

عملگرها 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.

انواع عملگرها:

²⁴ Operators

عمل‌گرهای ریاضی (حسابی)^{۲۵} •

عمل‌گرهای مقایسه‌ای^{۲۶} •

عمل‌گرهای تخصیص (انتسابی)^{۲۷} •

عمل‌گرهای منطقی^{۲۸} •

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** اشاره خواهد شد.

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
ضرب (*)	برای ضرب کردن دو یا چند عدد استفاده می‌شود.	Result = 5 * 20 print(result) # Output: 100
تقسیم (/)	برای تقسیم کردن یک عدد بر عدد دیگر استفاده می‌شود.	Result = 50 / 2 print(result) # Output: 25

²⁵ Arithmetic Operators

²⁶ Comparison Operators

²⁷ Assignment Operators

²⁸ Logical Operators

Result = 17 // 4 print(result) # Output: 4	برای تقسیم کردن دو عدد و نادیده گرفتن باقیمانده استفاده می‌شود(خارج قسمت تقسیم).	تقسیم صحیح (//)
result = 2 ** 3 print(result) # Output: 8	به توان رساندن یک عدد	توان (**)
Result = 17 % 4 print(result) # Output: 1	برای بدست آوردن باقیمانده تقسیم دو عدد استفاده می‌شود.	باقیمانده (%)

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 عملگرهای بصورت بولین می‌باشد. این عملگرها در عبارات شرطی و ساختارهای کنترلی مانند if و while کاربرد دارند. در ادامه به انواع عملگرهای مقایسه‌ای در جدول ۴ اشاره خواهد شد.

مثال	توضیحات	نوع عملگر
x = 10 y = 15 result = x == y print(result) # Output: False	برای بررسی اینکه آیا دو مقدار با یکدیگر برابر هستند یا خیر، استفاده می‌شود.	مساوی (==)
x = 45 y = 40 result = x != y print(result) # Output: True	برای بررسی اینکه آیا دو مقدار با یکدیگر برابر نیستند یا خیر، استفاده می‌شود.	غیرمساوی (!=)

<pre>x = 20 y = 10 result = x > y print(result) # Output: True</pre>	برای بررسی اینکه آیا یک مقدار از مقدار دیگر بزرگتر است یا خیر، استفاده می‌شود.	بزرگتر از ($>$)
<pre>x = 45 y = 2 result = x < y print(result) # Output: False</pre>	برای بررسی اینکه آیا یک مقدار از مقدار دیگر کوچکتر است یا خیر، استفاده می‌شود.	کوچکتر از ($<$)
<pre>x = 56 y = 79 result = x >= y print(result) # Output: False</pre>	برای بررسی اینکه آیا یک مقدار از مقدار دیگر بزرگتر است یا با آن برابر است، استفاده می‌شود.	بزرگتر یا مساوی (\geq)
<pre>x = 9 y = 79 result = x <= y print(result) # Output: True</pre>	برای بررسی اینکه آیا یک مقدار از مقدار دیگر کوچکتر است یا با آن برابر است، استفاده می‌شود.	کوچکتر یا مساوی (\leq)

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 ۵ به عملگرهای تخصیص اشاره شده است.

مثال	توضیحات مثال	نوع عملگر

age = 25 print(age) # Output: 25	مقدار age برابر ۲۵ می‌باشد و همین مقدار را در خروجی چاپ کرده است.	عملگر تخصیص ساده (=)
age = 25 age += 5 print(age) # Output: 30	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 age را چاپ کرده است و خروجی برابر ۳۰ شده است.	عملگر تخصیص با جمع (=+)
age = 25 age -= 5 print(age) # Output: 20	در خط اول مقدار age برابر ۲۵ می‌باشد، در خط دوم مقدار age را با ۵ تفریق کرده و در متغیر age قرار داده و در خط سوم age را چاپ کرده است و خروجی برابر ۲۰ شده است.	عملگر تخصیص با ضرب (*=)
age = 25 age /= 5 print(age) # Output: 5	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 age را چاپ کرده است و خروجی برابر ۵ شده است.	عملگر تخصیص با تقسیم (/=)
age = 25 age %= 5 print(age) # Output: 0	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 age را چاپ کرده است و خروجی برابر ۰ شده است. چون باقیمانده ۲۵/۵ برابر صفر است.	عملگر تخصیص با باقی‌مانده (/=)
age = 5 age **= 3 print(age) # Output: 125	در خط اول مقدار age برابر ۵ می‌باشد، در خط دوم مقدار age را به ۳ رسانده و حاصل را در age قرار داده و در خط سوم age را چاپ کرده است و خروجی برابر ۱۲۵ شده است.	عملگر تخصیص با توان (**=)
age = 4 age //= 3 print(age) # Output: 8	در خط اول مقدار age برابر ۴ می‌باشد، در خط دوم خارج قسمت تقسیم age بر ۳ در متغیر age قرار داده و در خط سوم age را چاپ کرده است و خروجی برابر ۸ شده است.	عملگر تخصیص با تقسیم صحیح (//=)

۴-۴-۳ عمل‌گرهای منطقی

عمل‌گرهای منطقی در پایتون برای ترکیب عبارات شرطی (در بخش‌های بعدی توضیح داده خواهد شد) و ایجاد عبارات پیچیده‌تر استفاده می‌شوند. این عمل‌گرها 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** ۶ اشاره می‌شود.

نوع عمل‌گر	توضیحات	مثال
(and)	برای بررسی اینکه آیا هر دو عبارت درست هستند یا خیر، استفاده می‌شود. اگر هر دو عبارت درست باشد، True بر می‌گرداند.	Age = 20 Car_Certificate = True if age >= 18 and Car_Certificate: print("You can Buy Car")
(or)	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.")
(not)	برای ضد کردن نتیجه یک عبارت استفاده می‌شود.	Is_raining = False if not is_raining: print("Today is Sunny.")

دو عبارت A و B را در نظر بگیرید:

- برای عمل‌گر منطقی و (and):

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

(False) نادرست	(True) درست	(False) نادرست
(False) نادرست	(False) نادرست	(False) نادرست

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
```

مثالی از عملگر منطقی and

خروجی:

True

False

False

False

برای عملگر منطقی یا (or) :

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

(True) درست	(True) درست	(False) نادرست
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)
```

مثالی از عملگر منطقی or

خروجی:

True

True

True

False

۳-۴-۵) اولویت عملگرها

عملگرها	عملگرها	عملگرها	اولویت
به توان رساندن یک عدد با عددی دیگر	توان (**)		۱
ضرب و تقسیم از چپ به راست انجام می‌شوند	ضرب (*) و تقسیم (/)		۲

جمع و تفریق از چپ به راست انجام می‌شوند	جمع (+) و تفریق (-)	۳
مقایسه دو مقدار با یکدیگر	مقایسه $(==, !=, <, >, <=, >=)$	۴
. نسبت به or and not نسبت به or اولویت دارد. and نسبت به or اولویت دارد.	منطقی (not, and, or)	۵
انتساب مقادیر به متغیرها	تخصیص(انتسابی) $(//=, \%=, /=, =*, -=, =+, =)$	۶

۵-۳) معرفی توابع وابسته

در برنامه‌نویسی شیء‌گرا، اشیاء شامل داده‌ها و رفتارهایی هستند که این داده‌ها را مدیریت می‌کنند. این رفتارها در قالب توابع وابسته یا همان متدها^{۲۹} 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.

²⁹ Methods

فصل چهارم

متغیرها و انواع داده‌ها در زبان برنامه‌نویسی پایتون

۴) متغیرها و انواع داده‌ها در زبان برنامه‌نویسی پایتون

۱-۴) متغیرها^{۳۰} در پایتون

متغیرها 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 می‌کنیم. به این ترتیب، نام متغیر را در سمت چپ و مقدار متغیر را بعد از علامت مساوی، در سمت راست قرار داده می‌شود (شکل).

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 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 ۲ انواع مختلفی از داده‌هایی که در پایتون وجود دارد.

³⁰ Variables

نوع داده	توضیحات	مثال
متغیرهای عددی صحیح (int)	برای ذخیره اعداد بدون اعشار	50, 20, 23
متغیرهای عددی اعشاری (float)	برای ذخیره اعداد با اعشار	40.55, 23.64, 33.2, 26.0001
متغیرهای 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")
لیست (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 تشخیص نوع داده‌ی یک متغیر از دستور زیر استفاده می‌شود:

print(type(variable Name))

python example - Numbers.py

```
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)}")
```

مثالی از دستور `type` در پایتون

خروجی:

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^{۳۱} در پایتون

۴-۲-۱) اعداد در پایتون

در ساختار پایتون می‌توان سه نوع عدد را تعریف کرد:

- int

^{۳۱} Data Types

- float
- complex

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

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

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)}")
```

مثالی از نوع عددی int در پایتون

خروجی:

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)}")
```

مثالی از نوع عددی float در پایتون

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)})
```

مثالی از نوع عددی `complex` در پایتون

خروجی:

```
Type of (10+5j) is <class 'complex'>
Type of (-0-25j) is <class 'complex'>
Type of 1j is <class 'complex'>
```

۴-۲-۲) رشته‌ها^{۳۲} در پایتون

در پایتون، نوع داده رشته، یک توالی از کاراکترها است به عبارت دیگر به تعدادی کاراکتر گفته می‌شود که داخل یک جفت کوئیشن(“”) یا دابل کوئیشن(“”) قرار بگیرند.

³² Strings

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)}")
```

مثالی از تعریف کردن رشته‌ها در پایتون

خروجی:

```
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

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)	باتوجه 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 value 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: یک عدد صحیح است که جستجو را از آن موقعیت شروع می‌کند و مقدار end: پیش فرض برابر ۰ است.

<p>در این تابع وابسته، با استفاده از مقدار <code>tabsize</code> که یک عدد صحیح می‌باشد تعداد فاصله هر <code>tab</code> در رشته مشخص می‌شود. یعنی در هر <code>tab</code> چه تعداد فضای خالی ایجاد می‌شود (پیش فرض مقدار آن ۸ است).</p>	<code>expandtabs(tabsize)</code>
<p>رشته را برای یک مقدار <code>value</code> جستجو می‌کند و موقعیت جایی که <code>value</code> پیدا شده است را برمی‌گرداند.</p> <p><code>start</code> : جستجو را از کجا شروع شود. مقدار پیش فرض برابر <code>0</code> است.</p> <p><code>end</code>: جستجو کجا پایان یابد. پیش فرض انتهای رشته است.</p> <p>رشته را با توجه به مقدار <code>value</code> 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 <code>find()</code> اولین مقداری که از سمت چپ به راست پیدا می‌کند، موقعیت آن را برمی‌گرداند اما تابع وابسته <code>rfind()</code> آخرین مقداری که از سمت چپ به راست پیدا می‌کند، موقعیت آن را برمی‌گرداند.</p> <p><code>start</code> : جستجو را از کجا شروع شود. مقدار پیش فرض برابر <code>0</code> است.</p> <p><code>end</code>: جستجو کجا پایان یابد. پیش فرض انتهای رشته است.</p>	<code>find(value, start, end)</code> <code>rfind(value, start, end)</code>
<p>این تابع وابسته مقدارهای مشخص شده را قالب بندی می‌کند و آن‌ها را دورن نگهدارنده‌های متغیر مربوطه قرار می‌دهد.</p>	<code>format(value1, value2...)</code>
<p>با توجه به مقدار <code>value</code> که می‌گیرد، آن <code>value</code> را در رشته جستجو می‌کند و اگر پیدا کند موقعیت آن <code>value</code> در رشته را برمی‌گرداند.</p> <p><code>start</code> : جستجو را از کجا شروع شود. مقدار پیش فرض برابر <code>0</code> است.</p> <p><code>end</code>: جستجو کجا پایان یابد. پیش فرض انتهای رشته است.</p>	<code>index(value, start, end)</code>
<p>رشته را برای یک مقدار مشخص جستجو می‌کند و آخرین موقعیت جایی که پیدا شده را برمی‌گرداند. با توجه به مقداری که می‌گیرد، اگر آن مقدار در یک رشته تکرار شده باشد، تابع وابسته <code>index()</code> اولین مقداری که از سمت چپ به راست پیدا</p>	<code>rindex(value, start, end)</code>

می کند، موقعیت آن را برمی گرداند اما تابع وابسته `rindex()` آخرین مقداری که از سمت چپ به راست پیدا می کند، موقعیت آن را برمی گرداند. تفاوتی که `rindex` با `rfind` دارد این است که اگر مقدار در رشته وجود نداشته باشد، `rfind` مقدار ۱- را برمی گرداند اما `rindex` ارور می دهد.

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

`:end`: جستجو کجا پایان یابد. پیش فرض انتهای رشته است.

اگر همه کarakترهای رشته به صورت عددی و الفبایی باشند، `True` را برمی گرداند.
اگر در رشته ها حروف (`space, !, #, %, &, ?, ...`) باشد مقدار `False`

برمی گرداند.

`isalnum()`

اگر همه کarakترهای رشته به صورت کاراکتر الفبا باشند، `True` را برمی گرداند.

`isalpha()`

اگر همه کarakترهای رشته کarakترهای `ascii` باشند، `True` را برمی گرداند.

`isascii()`

اگر همه کarakترهای رشته به صورت عددی و رقم باشند، `True` را برمی گرداند.

`isdigit()`

اگر تمام کarakترهای رشته با حروف کوچک باشند، `True` را برمی گرداند.

`islower()`

اگر همه کarakترهای رشته به صورت عددی و رقم باشند، `True` را برمی گرداند.

تفاوت اصلی بین دو تابع `isnumeric()` و `isdigit()` این است: تابع وابسته

`isnumeric()`

`isnumeric()` اعداد رومی را هم به عنوان عدد و رقم می شناسد.

اگر همه کarakترهای رشته `space` یا فاصله باشند، `True` را برمی گرداند.

`isspace()`

اگر فقط یک کarakter `space` نباشد، مقدار `False` برمی گرداند.

اگر رشته از قوانین `title` پیروی کند، `True` را برمی گرداند.

`istitle()`

<p>اگر همه کلمات در یک متن با حروف بزرگ شروع شوند، و بقیه کلمات حروف کوچک هستند، در غیر این صورت False</p>	
<p>اگر همه کاراکترهای رشته با حروف بزرگ باشند، True را برمی‌گرداند</p>	isupper()
<p>یک مقدار iterable مثل لیست یا توپل یا ... می‌گیرد و به یک رشته تبدیل می‌کند.</p>	join(iterable)
<p>باتوجه 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.</p>	strip(characters)
<p>باتوجه 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.</p>	lstrip(characters)
<p>باتوجه 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.</p>	rstrip(characters)
<p>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.</p>	partition(value)
<p>این تابع وابسته برای جایگزینی یک مقدار با مقدار جدید استفاده می‌شود و دو مقدار می‌گیرد که مقدار اول همان oldValue است که می‌خواهید حذف و جایگزین کنید و مقدار دوم newValue است که می‌خواهید جایگزین کنید. count: اختیاری است. بصورت عددی می‌باشد. چند مورد از مقدار قدیمی را می‌خواهید جایگزین کنید. پیش فرض همه موارد است.</p>	replace(oldValue, newValue, count)

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 لیست برمی‌گرداند.

هر دو مقدار ورودی، اختیاری است.

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

(white space) مقدار پیش‌فرض : “ ” 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 لیست برمی‌گرداند.

تنها تفاوت بین split() و rsplit() استفاده مقدار maxsplit که به داده 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.

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

(white space) مقدار پیش‌فرض : “ ” separator

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

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 start پیش‌فرض برابر ۰ است.

split(separator, maxsplit)

rsplit(separator, maxsplit)

splitlines()

startswith(value, start, end)

: 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 start پیش فرض برابر ۰ است.

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

endswith(value, start, end)

۲-۲-۴ مفاهیم slicing و indexing و concatenation

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}")
```

مثالی از مفهوم concentration در پایتون

خروجی:

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

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

مفهوم indexing در پایتون

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

برای استفاده معمولا 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}")
```

مثالی از مفهوم indexing در پایتون

خروجی:

```
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
```

مفهوم slicing در پایتون

در پایتون، slicing (برش) به شما این امکان را می‌دهد که بخشی از یک رشته (یا لیست، 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 می‌یابد (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 می‌یابد 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 ابتدای ایندکس مشخصی (برای مثال تا ایندکس دهم):
`x = string[:10]`
- برش رشته از یک ایندکس مشخصی تا آخر (برای مثال برش از ایندکس سوم):
`x = string[3:]`
- ایندکس گذاری منفی (به مثال 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}")
```

مثالی از مفهوم slicing در پایتون

خروجی:

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.

• **upper()** تابع وابسته

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}")
```

مثالی از توابع وابسته‌ی Capitalize و Upper و Lower

خروجی:

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 casfold()**

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 `casfold` برای مقایسه رشته‌ها در زبان‌هایی که دارای تفاوت‌های ظریف 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.

```
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}")
```

مثالی از توابع وابسته‌ی `title`, `casefold` و `swapcase`

خروجی:

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}")
```

مثالی از تابع وابسته strip

خروجی:

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 ۱۰ 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.

python example - strings.py

```
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}")
```

مثالی از تابع وابسته Replace() در پایتون

خروجی:

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.

معمولاً زمانی که از این تابع وابسته برای جداسازی یک رشته استفاده می‌کنید، مقداری که برمی‌گرداند، یک لیست 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 لیست 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}")
```

مثالی از تابع وابسته `split`

خروجی:

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

تابع وابسته `find()` •

تابع وابسته `index()` •

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، مقدار است که آن مقدار را در رشته جستجو می‌کند. مقدار دوم مشخص می‌کند که فرایند جستجو در رشته از چه ایندکسی شروع شود. مقدار سوم نیز مشخص می‌کند فرایند جستجو در کدام ایندکس پایان یابد(خود ایندکس 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 می‌باشد. اگر تابع وابسته `find` مقدار داده شده را در رشته پیدا نکرد، نتیجه‌های که برمی‌گرداند ۱ - می‌باشد، اما اگر تابع وابسته `index` مقدار داده شده در رشته را پیدا نکرد، خطأ و ارور 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")}")
```

مثالی از تابع وابسته `find`

خروجی:

*location of 'fruits' is: 21
location of 'Apples' is: 0*

location of 'orange' is: -1

python example - strings.py

```
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')}")
```

مثالی از تابع وابسته `index`

خروجی:

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

```
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)}")
```

مثالی از توابع وابسته‌ی `find` و `index` با اعمال مقادیر دوم و سوم هر تابع وابسته

خروجی:

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

۴-۲-۳) لیست‌ها (فهرست‌ها)^{۳۳} در پایتون

لیست‌ها در پایتون برای نگهداری انواع داده‌ها استفاده می‌شود به عبارت دیگر نوعی مخزن است که در آن انواع داده‌ها قرار می‌گیرد.

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

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

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

متداول‌ترین روش ایجاد لیست در پایتون، با استفاده از کروشه می‌باشد، اما می‌توان از توابع داخلی که در آینده پرداخته می‌شود، استفاده کرد.

- استفاده از کروشه([[]])
- استفاده از تابع داخلی list

ساختار کلی لیست‌ها در پایتون:

Custom_List_Name = [item1, item2, ...]

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

^{۳۳} Lists

```
1 # Lists in Python ( فهرست‌ها در پایتون )
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]]
```

مثال‌هایی از ساختار فهرست‌ها در پایتون

داده‌هایی که در لیست‌ها می‌توان قرار داد:

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

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

مثال	توضیحات	تابع وابسته
<pre>X = ["Reza", "Ali", "Mahdi"] X.append("Hossein") print(X) # ["Reza", "Ali", "Mahdi", "Hossein"]</pre>	<p>برای اضافه کردن فقط یک داده به آخر لیست استفاده می‌شود.</p> <p>یک مقدار می‌گیرد.</p>	append(value)
<pre>X = ["Reza", "Ali", "Mahdi"] X.clear() print(X) # []</pre>	<p>برای حذف کردن تمامی داده‌های لیست استفاده می‌شود.</p> <p>مقداری نمی‌گیرد.</p>	clear()
<pre>X = ["Reza", "Ali", "Mahdi"] Y = X.copy() print(Y) # ["Reza", "Ali", "Mahdi"]</pre>	<p>یک کپی از لیست گرفته و مقدار کپی شده را بر می‌گرداند.</p> <p>مقداری نمی‌گیرد.</p>	copy(list)
<pre>X = ["Reza", "Ali", "Mahdi", "Ali"] N = X.count("Ali") print(N) # 2</pre>	<p>تعداد دفعات تکرار شده را با توجه به مقداری که گرفته، بر می‌گرداند.</p> <p>یک مقدار می‌گیرد.</p>	count(value)
<u>برای اضافه کردن چندین داده به آخر لیست</u>		
<pre>X = ["Reza", "Ali", "Mahdi"] Y = ["Hossein", "Abolfazl"] X.extend(Y) print(X) # ["Reza", "Ali", "Mahdi", "Hossein", "Abolfazl"]</pre>	<p>استفاده می‌شود.</p> <p>یک مقدار می‌گیرد.</p> <p>نکته: داده‌ها حتما باید در داخل یک لیست یا توپل که قابلیت پیمایش دارد، قرار گیرد تا عملیات اضافه شدن صورت گیرد.</p>	extend(list)
<pre>X = ["Reza", "Ali", "Mahdi"] M = X.index("Reza")</pre>	<p>ایندکس مقداری که گرفته، بر می‌گرداند.</p>	index(value)

print(M)	یک مقدار می‌گیرد.
# 0	
X = ["Reza", "Ali", "Mahdi"]	از لیست.
Y = X.insert(1, "Hossein")	دو مقدار می‌گیرد. مقدار اول مربوط به مکانی insert(index, value)
print(Y)	است که می‌خواهید داده در آن مکان قرار
# ["Reza", "Hossein", "Ali", "Mahdi"]	گیرد و معمولاً با ایندکس مشخص می‌شود. و
	مقدار دوم داده‌ای است که اضافه می‌شود.
	برای حذف کردن یک داده مشخص از لیست
X = ["Reza", "Ali", "Mahdi"]	استفاده می‌شود.
X.pop(2)	یک مقدار می‌گیرد و آن هم <u>ایندکس مقداری</u> pop(index)
print(X)	که می‌خواهید حذف شود.
# ["Reza", "Ali"]	
	برای حذف کردن یک داده مشخص از لیست
X = ["Reza", "Ali", "Mahdi"]	استفاده می‌شود.
X.remove("Ali")	یک مقدار می‌گیرد و آن هم <u>خود مقداری</u> که remove(value)
print(X)	می‌خواهید حذف شود.
# ["Reza", "Mahdi"]	
X = ["Reza", "Ali", "Mahdi", "Hossein"]	ترتیب لیست را معکوس می‌نماید.
X.reverse()	reverse()
print(X)	مقداری نمی‌گیرد.
# ["Hossein", "Mahdi", "Ali", "Reza"]	
X = ["Reza", "Ali", "Mahdi", "Hossein"]	برای مرتب سازی لیست استفاده می‌شود.
Y = [5, 9, 25, 95, 200, 1]	دو مقدار اختیاری می‌گیرد.
print(X)	sort(reverse, key)
# ["Ali", "Hossein", "Mahdi", "Reza"]	مقدار اول True یا False است. اگر مقدار
	بگیرد، فرایند مرتب‌سازی را به صورت True

```

print(Y)                               معکوس و از بزرگ به کوچک انجام می‌دهد و
# [1, 5, 9, 25, 95, 200]               اگر False بگیرد که مقدار پیش‌فرض
                                         می‌باشد، فرایند مرتب‌سازی را از کوچک به
                                         بزرگ انجام می‌دهد.

                                         مقدار دوم، مقداری است که چگونگی
                                         مرتب‌سازی را از کاربر می‌گیرد. عموماً این
                                         مقدار توسط توابع lambda صورت می‌گیرد.

```

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

برای اینکه تعداد عناصر یک لیست 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 # Lists in Python (فهرست‌ها در پایتون)
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)}")

```

به دست آوردن تعداد اعضای لیست با استفاده از تابع `len()`

خروجی:

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 ۳، اعضای درون لیست، از لیست‌های مختلف تشکیل شده است، بنابراین تعداد لیست‌های تشکیل شده مهم هستند و اعضای داخل لیست‌ها در این قسمت مهم نیستند، چراکه تعداد اعضای لیست اصلی را حساب می‌کند و اعضای داخل لیست‌ها را محاسبه نمی‌کند.

برای محاسبه اعضای داخل لیست‌ها، باید تک به تک لیست‌های موجود در داخل لیست اصلی را استخراج کرده و تعداد اعضای هر لیست را توسط تابع `len` محاسبه کنید.

۴-۳-۲-۴) دسترسی به عناصر لیست

برای این‌که یک عنصر خاصی از یک لیست 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)
```

دسترسی به یک عنصر خاص لیست

خروجی:

1364

Yellow

هم‌چنین می‌توانید یک محدوده مشخص از لیست را انتخاب کرده و در یک لیست 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}')
```

دسترسی به محدوده خاص از لیست

خروجی:

```
select_red_to_yellow: ['Red', 98536, 'Yellow']  
select_custom: [20, 1364, 1, '2416']
```

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

۴-۲-۳-۵) تغییر عناصر یک لیست

برای تغییر یک عنصر خاص از یک لیست می‌توانید با گرفتن ایندکس 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}")
```

مثالی از تغییر یکی از عناصر لیست

خروجی:

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

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

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

```
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}")
```

تغییر چند عنصر از لیست

خروجی:

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 1-EndIndex پایان می‌یابد.

۴-۳-۲-۶) افزودن عناصر جدید به لیست

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

- استفاده از تابع وابسته append برای افزودن فقط یک عنصر به انتهای لیست.

استفاده از تابع وابسته `insert` برای افزودن یک عنصر به مکان دلخواه.

استفاده از تابع وابسته `extend` برای افزودن مجموعه‌ای از عناصر به انتهای لیست.

python example - lists.py

```
93 # Lists in Python ( فهرست‌ها در پایتون )
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
```

افزودن یک عنصر جدید به لیست با استفاده از توابع وابسته‌ی `append` و `insert`

خروجی:

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']

```
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}")
```

افزودن عناصر جدید به لیست با تابع وابسته `extend`

خروجی:

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

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

باتوجه به مثال شکل ۹، طبق خط ۱۱۶ از کد، شما می‌توانید، به دو صورت از تابع وابسته `extend` استفاده کنید:

- ✓ یک لیست جداگانه ایجاد کرده و نام آن لیست را به تابع وابسته `extend` دهید.
- ✓ داخل تابع وابسته `extend` لیست ایجاد کرده و عناصری که می‌خواهید اضافه کنید، در داخل لیست قرار دهید.

۷-۳-۲-۴) حذف عنصر/عناصر یک لیست

از روش‌های زیر برای حذف عناصر لیست استفاده می‌شود:

- استفاده از تابع وابسته `remove` برای حذف یک عنصر با استفاده از نام آن عنصر.
- استفاده از تابع وابسته `pop` برای حذف یک عنصر با استفاده از ایندکس آن عنصر.
- استفاده از کلیدواژه `del` برای حذف یک عنصر خاص (`del list_name[index]`).
- استفاده از تابع وابسته `clear` برای حذف تمامی عناصر یک لیست.

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}")
```

حذف عنصر با توابع وابسته `pop` و کلیدواژه `del`

خروجی:

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}")
```

حذف تمامی عناصر با تابع وابسته clear

Car List:

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

Remove all items from Car List by clear():

`[]`

۴-۲-۳-۸) کپی گیری از لیست

دو روش برای کپی گیری از لیست 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}")
```

مثالی از کپی گیری از فهرست با دو روش

خروجی:

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) لیست‌های تو در تو

لیست‌های تو در تو^{۳۴} یا همان لیست‌های چندسطحی، یک ساختار داده پیشرفته در پایتون هستند که امکان ذخیره‌سازی لیست‌ها به عنوان عناصر درون یک لیست دیگر را فراهم می‌کنند. این ساختار برای نمایش داده‌های پیچیده مانند ماتریس‌ها، جداول، یا روابط سلسله‌مراتبی بسیار مفید و منظم است.

برای ایجاد لیست‌های تو در تو، می‌توانید یک یا چند لیست را به عنوان عناصر داخل یک لیست 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)
```

مثالی از فهرست‌های تو در تو

خروجی:

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

دسترسی به عناصر لیست‌های تو در تو

³⁴ Nested Lists

برای دسترسی به عناصر لیست‌های تو در تو، 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
```

خروجی:

[1, 2, 3]

6

افزودن و حذف عناصر در لیست‌های تو در تو

مانند لیست‌های عادی، می‌توانید عناصر جدیدی به لیست‌های تو در تو اضافه کنید یا آن‌ها را حذف نمایید.

مثال برای افزودن عنصر:

```

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}")

```

خروجی:

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')]

مثال برای حذف عنصر:

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}")
```

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]]

۴-۳-۱۰) ادغام کردن دو لیست

برای ادغام کردن دو لیست در یک لیست دو روش وجود دارد:

استفاده از حلقه‌ها (در آینده توضیح داده خواهد شد). •

استفاده از تابع وابسته `List_1.extend(List_2)` •

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}")
```

ادغام کردن دو لیست با حلقه for و تابع وابسته `extend`

خروجی:

```
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^{۳۵} 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 یک فرم وجود دارد که کاربر باید آن را تکمیل کند، مقادیری که کاربر باید آن‌ها را تکمیل کند شامل نام و نام خانوادگی، کد ملی، سن، رنگ مورد علاقه و موارد دیگر است. معمولاً باید همه‌ی این داده‌ها را ذخیره کرد. روش اول برای ذخیره‌سازی به این صورت است که برای هر جواب کاربر، یک متغیر جداگانه ایجاد کرد، ولی این روش بسیار طولانی و از طفی منجر به افزایش خطای در کدنویسی می‌شود. روش دوم استفاده از یک لیست یا فهرست برای پاسخ‌های کاربر است، اما ممکن است که کاربر رنگ مورد علاقه خود را ننویسد و یا این‌که کاربر

³⁵ Dictionaries(Dictionary)

نام خود را در انتهای یک لیست نوشته و قرار دهد، این کار احتمال خطأ را افزایش می‌دهد. زبان برنامه‌نویسی پایتون برای کنترل این موقعیت‌ها و دسته‌بندی داده‌ها یک نوع داده‌ای با نام دیکشنری معرفی کرده است.

۱-۴-۲-۴) تعریف و ایجاد دیکشنری‌ها در پایتون

دیکشنری‌ها معمولاً با مقادیر جفت کلید^{۳۶} و مقدار^{۳۷} 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 (در آینده توضیح داده خواهد شد.)

ساختار کلی دیکشنری‌ها در پایتون

Custom_Dictionar_Name = {key1:value1, key2:value2}

در شکل ۱۴، مثال‌هایی از دیکشنری‌ها در پایتون آورده شده است.

³⁶ Key

³⁷ Value

```
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 }
```

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(رشته، عدد، لیست، مجموعه‌ها، تاپل‌ها).
حتی می‌توان یک دیکشنری دیگری را عنوان مقدار یک کلید تعریف کرد.
- ✓ برای مقادیر کلیدها می‌توانید تابع نیز تعریف کنید به عبارت دیگر یک تابع به عنوان مقدار برای کلید بدهید(در بخش‌های بعدی توابع گفته می‌شود و این ویژگی دیکشنری نیز گفته خواهد شد).

✓ معمولا 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"]()
45 print(x)
```

تعريف تابع داخل دیکشنری و استفاده از تابع

خروجی:

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

تابع وابسته	توضیحات	مثال
clear()	پاک کردن تمامی عناصر(کلید و مقدار) دیکشنری مقداری نمی‌گیرد.	<pre>Car = { "brand": "Corvette", "model": "C8", "year": 2023 } Car.clear() print(Car) # {}</pre>
copy()	یک کپی از دیکشنری گرفته و مقدار کپی شده را برمی‌گرداند.	<pre>Car = { "brand": "Corvette", "model": "C8", "year": 2023 } X = Car.copy() print(X) # { "brand": "Corvette", "model": "C8", "year": 2023 }</pre>
fromkeys(keys, values)	با توجه به مقادیری 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 List یا ... که قابلیت پیمایش iterable بر روی آیتم‌ها را دارد، مقداردهی کرد.	<pre>X = ("brand", "model", "year") Y = ("Camaro", "ZL1", 2021) New_dict = {}.fromkeys(X, Y) print(New_dict) # { "brand": "Camaro", "model": "ZL1", "year": 2021 }</pre>

مقدار دوم اختیاری است و مربوط به مقادیر هر کلید می‌باشد و باید در قالب Tuple یا List یا... که قابلیت پیمایش بر روی آیتم‌ها را دارد، مقداردهی کرد.

```
Car = {
    "brand": "Corvette",
    "model": "C8",
    "year": 2023
}
X = Car.get("model")
print(X)
# C8
```

باتوجه به مقداری که می‌گیرد، آن مقدار را در کلیدهای دیکشنری جستجو کرده و اگر کلیدی با همان مقداری که گرفته وجود داشته باشد، مقدار get(keyName) آن کلید را برمی‌گرداند. یک مقدار می‌گیرد.

```
Car = {
    "brand": "Corvette",
    "model": "C8",
    "year": 2023
}
X = Car.items()
print(X)
# [("brand", "Corvette"),
# ("model", "C8"), ("year", 2023)]
```

تمامی اجزای موجود در دیکشنری اعم از کلید و مقدار را در داخل لیست به صورت (key,value) items() برمی‌گرداند. مقداری نمی‌گیرد.

```
Car = {
    "brand": "Corvette",
    "model": "C8",
    "year": 2023
}
X = Car.keys()
print(X)
# ["brand", "model", "year"]
```

کلیدهای موجود در داخل دیکشنری را در داخل یک لیست برمی‌گرداند. keys() مقداری نمی‌گیرد.

```
Car = {
    "brand": "Corvette",
    "model": "C8",
```

باتوجه به مقداری که می‌گیرد، آن مقدار را در کلیدهای دیکشنری جستجو کرده و اگر کلیدی pop(keyName)

```

"year": 2023
}
Car.pop("model")
print(Car)
# {
"brand": "Corvette",
"year": 2023
}

```

همانند مقداری که داده شده پیدا کند، کلید و مقدار آن را در دیکشنری پاک می کند.

یک مقدار می گیرد.

```

Car = {
"brand": "Corvette",
"model": "C8",
"year": 2023
}
Car.popitem()
print(Car)
# {
"brand": "Corvette",
"model": "C8"
}

```

آخرین کلید و مقدار در دیکشنری را پاک می کند.

popitem()

مقداری نمی گیرد.

```

Car = {
"brand": "Corvette",
"model": "C8",
"year": 2023
}
X = Car.setdefault("year")
print(X)
# 2023

```

باقیه به مقداری که می گیرد، آن مقدار را در کلیدهای دیکشنری جستجو کرده، اگر موجود باشد، مقدار آن کلید را در دیکشنری برمی گرداند.

اگر کلید مشخص شده وجود نداشته باشد یک مقدار از پیش تعريف شده را برمی گرداند.

setdefault(keyName, value)

مقدار اول مقداری است در کلیدهای دیکشنری جستجو می کند.

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.

```
Car = {
    "brand": "Corvette",
    "model": "C8",
    "year": 2023
}
Car.update({"color": "Yellow"})
print(Car)
```

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

```
# {
    "brand": "Corvette",
    "model": "C8",
    "year": 2023,
    "color": "Yellow"
}
```

```
Car = {
    "brand": "Corvette",
    "model": "C8",
    "year": 2023
}
X = Car.values()
print(X)
# ["Corvette", "C8", 2023]
```

بر روی تمامی کلیدهای دیکشنری پیمایش کرده و مقدارشان را در داخل لیست قرار داده و آن لیست را برمی‌گرداند.

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، نوشته می‌شود. سپس مقدار آن کلید را در قسمت `value` نوشته 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}")
```

مثالی از تعریف کلید و مقدار برای دیکشنری در پایتون

خروجی:

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']}

۴-۴-۲-۴) تغییر مقدار یک کلید در دیکشنری

برای این که 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}")
```

مثالی از تغییر مقدار یک کلید دیکشنری

خروجی:

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'}

۴-۲-۵) حذف کلید و مقدار در دیکشنری

برای حذف کلید و مقدارش چند روش وجود دارد.

روش اول که مربوط به تابع وابسته `pop` میباشد که در قسمت قبل توضیح داده شد.

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 پاک کردن تمامی عناصر اعم از کلید و مقدار در دیکشنری

```
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}")
```

مثالی از حذف کلید و مقدار از دیکشنری با استفاده از توابع وابسته‌ی `pop` و `popitem` و کلیدواژه `del`

خروجی:

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'}`

```
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
```

مثالی از حذف تمامی کلید و مقدارها با استفاده از تابع وابسته `clear` در دیکشنری

خروجی:

Before using clear method:

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

After using clear method:

{}

۴-۲-۶) کپی‌گیری 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 ادامه توضیح مختصری داده خواهد شد و در آینده توضیحات کامل ارائه 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}")
```

مثالی از کپی‌گیری از دیکشنری با تابع وابسته `copy` و تابع داخلی `dict`

خروجی:

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}

۷-۴-۲-۴) دیکشنری‌های تو در تو

دیکشنری‌های تو در تو^{۳۸} 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)
```

خروجی:

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

³⁸ Nested Dictionaries

}

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.

مثال:

python example - dictionaris.py

```
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']}")
```

خروجی:

Person1 info is:

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

Name of person2 is:

Ali

Age of person3 is:

24

افزودن و بهروزرسانی در دیکشنری‌های تو در تو

شما می‌توانید به راحتی کلیدها و مقادیر جدیدی به دیکشنری‌های تو در تو اضافه کنید یا مقادیر موجود را تغییر دهید.

مثال:

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}")
```

خروجی:

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}}
```

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}")
```

خروجی:

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}}
```

۵-۲-۴) تاپل‌ها در پایتون

تاپل‌ها^{۳۹} برای ذخیره چندین عنصر(داده) در یک متغیر استفاده می‌شوند. نوع داده تاپل‌ها در پایتون شبیه لیست‌ها می‌باشد با این تفاوت که تاپل‌ها تغییرناپذیرند.^{۴۰} و نمی‌توان اعضای آن را تغییر دهید. از آنجا که تاپل‌ها تغییرناپذیر می‌باشند، دستورات حلقه با تاپل‌ها سریع‌تر از لیست‌ها 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 لیست‌ها دارد در این است که برای ایجاد لیست‌ها از کروشه یا براکت ([]) استفاده می‌شود اما برای تعریف و ایجاد تاپل‌ها از پرانترز (()) استفاده می‌شود. تاپل‌ها همانند لیست‌ها می‌توانند عناصر تکراری نیز داشته باشند.

به طور خلاصه ویژگی‌هایی که تاپل‌ها دارد عبارت است از:

³⁹ Tuples

⁴⁰ Immutable

- مرتب شده^{۴۱}: موارد دارای ترتیب مشخصی هستند و این ترتیب تغییر نمی‌کند.
- تغییرناپذیر^{۴۲}: پس از ایجاد تاپل، امکان ایجاد تغییر، حذف و اضافه در تاپل‌ها وجود ندارد.
- قابلیت تکراری بودن^{۴۳}: عناصری که در تاپل‌ها وجود دارد، می‌توانند تکراری نیز باشند.

۱-۵-۲-۴) 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 لیست یا تاپل یا داده‌ای که قابلیت پیمایش دارد، قرار گیرد).

python example - tuples.py

```

1 # تاپل در پایتون)
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)}")

```

⁴¹ Ordered

⁴² Unchangeable

⁴³ Allow Duplicates

تعریف و ایجاد تاپل‌ها در پایتون

خروجی:

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
- لیست‌ها
- دیکشنری‌ها
- مجموعه‌ها
- تاپل‌ها
- (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)
```

مثالی از قرار دادن انواع داده‌ها در تاپل‌ها

خروجی:

```
('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]})
```

۴-۵-۲-۴) تعداد عناصر موجود در تاپل‌ها

برای به دست آوردن تعداد عناصر یک تاپل می‌توان از تابع `len()` همانند لیست‌ها، 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)}")
```

به دست آوردن طول یک تاپل در پایتون

خروجی:

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، حتماً اطمینان حاصل کنید که بعد از عنصر موردنظر در داخل تاپل، کلاماً گذاشته شده است، در غیر این صورت نوع داده‌ای که تعریف کرده‌اید تاپل نیست. برای درک بهتر این موضوع به مثال زیر توجه کنید.

python example - tuples.py

```
55 # Tuples in Python (تایپ 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
```

مثالی از تایپ تک عنصری

خروجی:

```
tuple_example_2: ('Yellow',)
Type of tuple_example_2 is <class 'tuple'>
tuple_example_3: Yellow
Type of tuple_example_3 is <class 'str'>
```

۴-۵-۲-۳) توابع وابسته‌ی مربوط به تایپ‌ها در پایتون

تابع وابسته‌ی موجود در تایپ‌ها به دلیل غیرقابل تغییرپذیری بسیار اندک می‌باشد.

تابع وابسته	توضیحات	مثال
-------------	---------	------

<pre> Tuple_example = ("BMW", "Benz", "Ford", "Benz", "Ford", "Benz") X = Tuple_example.count("Benz") print(X) # 3 </pre>	بایوچه به مقداری که می‌گیرد، تعداد دفعاتی که آن مقدار در تاپل تکرار شده است را برمی‌گرداند. count(value) یک مقدار می‌گیرد.
<pre> Tuple_example = ("BMW", "Benz", "Ford", "Benz", "Ford", "Benz") X = Tuple_example.index("Ford") print(X) # 2 </pre>	بایوچه به مقداری که می‌گیرد، آن مقدار را در تاپل جستجو می‌کند و اولین مقداری که با مقدار مشخص شده پیدا کند، ایندکس آن مقدار را از تاپل برمی‌گرداند. Index(value) یک مقدار می‌گیرد.

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 ساختار زیر، به آن عنصر دست یابید:

```

70 # 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]}")

```

دسترسی به عناصر یک تاپل

خروجی:

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

-1 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 به مثال زیر استفاده 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}")
```

مثالی از ایندکس منفی در تاپل‌ها

خروجی:

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:

1253

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

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

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

python example - tuples.py

```
113 # Tuples in Python (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)}")
```

دسترسی به محدوده مشخص از تاپل

خروجی:

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 کلیدوازه `in` دارید. مقداری که

برمی‌گرداند، یک بولین می‌باشد و مقادیر `True` یا `False` برمی‌گرداند.

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

python example - tuples.py

```
132 # تاپل در پایتون (Tuples in Python)
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
```

بررسی موجود بودن یک عنصر در داخل یک تاپل

خروجی:

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

۴-۵-۲-۴) 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 لیست: ابتدا باید آن تاپل را به لیست تبدیل کرده و تغییر خود را اعم از افزودن، حذف و ... اعمال کرده و درنهایت دوباره تبدیل به تاپل کنید.

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

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

```
151 # Tuples in Python (تایپ در پایتون)  
152 # Change Element of Tuples (تغییر عناصر تایپ)  
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}")
```

تغییر عناصر تایپ با روش تبدیل تایپ به لیست

خروجی:

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}')
```

اضافه کردن یک تاپل به تاپل دیگر

خروجی:

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

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

۶-۲-۴) مجموعه‌ها در پایتون

مجموعه‌ها^{۴۴} نوع دیگری از انواع داده‌ها در پایتون می‌باشد که شبیه به مجموعه‌ها در دروس ریاضی می‌باشد.

مجموعه‌ها نیز همانند لیست‌ها و تاپل‌ها برای ذخیره چندین مقدار درون یک متغیر استفاده می‌شوند.

ویژگی‌های مجموعه‌ها

⁴⁴ Sets

- 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، امکان تغییرپذیری عناصر وجود ندارد. توجه داشته باشید که می‌توانید، عناصری به مجموعه اضافه و یا حذف کنید اما امکان 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 از دو روش زیر می‌توان استفاده کرد:

- روش اول، استفاده از علامت آکولاد (مشابه دیکشنری‌ها)
- روش دوم، استفاده از تابع داخلی set

```
sets.py
17 # Legal Data Types can use in Sets
18 set_example_3 = {"Reza", "Red", "Corvette", "1", "-0.4"} # Strings in Set
19 print(f"set_example_3: {set_example_3} and Type of set_example_3 is {type(set_example_3)}")
20
21 set_example_4 = {1, 2, 6, 8, 0.5, -523, 510.249, -7.8} # Numbers in Set
22 print(f"set_example_4: {set_example_4} and Type of set_example_4 is {type(set_example_4)}")
23
24 set_example_5 = {True, False} # Boolean in Set
25 print(f"set_example_5: {set_example_5} and Type of set_example_5 is {type(set_example_5)}")
26
27 set_example_6 = {"Reza", False, 23, 188.8, "Yellow", "1"} #
28 print(f"set_example_6: {set_example_6} and Type of set_example_6 is {type(set_example_6)}")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
set_example_3: {'1', 'Red', 'Reza', '-0.4', 'Corvette'} 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', '1', 23, 'Reza', 188.8} and Type of set_example_6 is <class 'set'>

نوع داده‌های مجاز برای استفاده در داخل مجموعه‌ها

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

برای به دست آوردن تعداد عناصر یک مجموعه می‌توان از تابع `(len)` همانند لیست‌ها و تاپل‌ها استفاده کرد.

```

sets.py
31 # Length of Sets
32
33 set_example_7 = {1, 2, 6, 8, 0.5, -523, 510.249, -7.8}
34 print(f"length of {set_example_7} is {len(set_example_7)}")
35
36 set_example_8 = {}
37 print(f"length of {set_example_8} is {len(set_example_8)}")
38
39 set_example_9 = {"Reza", False, 23, 188.8, "Yellow", "1"}
40 print(f"length of {set_example_9} is {len(set_example_9)}")
41
42 set_example_10 = {True}
43 print(f"length of {set_example_10} is {len(set_example_10)}")
44

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
length of {0.5, 1, 2, 6, 8, -523, -7.8, 510.249} is 8
length of {} is 0
length of {False, 'Yellow', '1', 23, 188.8, 'Reza'} is 6
length of {True} is 1
PS C:\Users\RezaS\Desktop\python example>

```

استفاده از تابع `len` جهت به دست آوردن تعداد عناصر مجموعه

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

تابع وابسته	توضیحات	مثال
<code>add(element)</code>	برای اضافه کردن یک عنصر به مجموعه استفاده می‌شود.	mySet = {'a', 'b', 'c'} mySet.add('d') print(mySet) # {'a', 'b', 'c', 'd'}
<code>clear()</code>	تمامی عناصر داخل مجموعه را حذف می‌کند. مقداری نمی‌گیرد.	mySet = {'a', 'b', 'c'} mySet.clear() print(mySet) # {}

```

mySet = {'a', 'b', 'c'}
newSet = mySet.copy()
print(newSet)
# {'a', 'b', 'c'}

```

از مجموعه کپی گیری کرده و مقدار کپی شده را برمی‌گرداند.

copy()

```

mySet = {'a', 'b', 'c'}
mySet.discard('a')
print(mySet)
# {'b', 'c'}

```

بایوچه 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.

discard(element)

```

mySet = {'a', 'b', 'c', 'd',
'e', 'f'}
mySet.pop()
print(mySet)
# {'a', 'b', 'c', 'e', 'f'}

```

یک مقداری را از مجموعه حذف می‌کند. مقداری نمی‌گیرد.

pop()

```

mySet = {'a', 'b', 'c'}
mySet.discard('c')
print(mySet)
# {'a', 'b'}

```

بایوچه به مقداری که می‌گیرد، آن را در مجموعه جستجو می‌کند و اگر پیدا کند، آن را حذف می‌کند. در غیر این صورت خطأ رور (Error) می‌دهد. یک مقدار می‌گیرد.

remove(item)

تفاضل مجموعه‌ها یا همان علامت تفریق (-) می‌باشد. بین دو یا چند مجموعه انجام می‌گیرد.

```

mainSet = {'a', 'b', 'c'}
set1 = {'c', 'd', 'e'}
result =
mainSet.difference( set1)
print(result)
# {'a', 'b'}

```

مقداری که برمی‌گرداند شامل عناصری است که فقط در مجموعه اول وجود دارد و در مجموعه‌های دیگر وجود ندارد.

مقدار برگردانده شده در یک مجموعه جدید قرار می‌گیرد.

**difference(set1, set2,
set3, ...)**

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'}
```

عنصری است که فقط در مجموعه اول وجود دارد و `difference_update(se`

```
mainSet.difference_update(se
```

در مجموعه‌های دیگر وجود ندارد.

```
e(set1)
```

مقدار برگردانده شده در مجموعه اولیه قرار می‌گیرد.

```
print(mainSet)
```

یک یا چندین مجموعه می‌گیرد.

```
# {'a', 'b'}
```

`difference_update(se`

`t1, set2, set3, ...)`

```
mainSet = {'a', 'b', 'c'}
```

نقش `&` را بازی می‌کند و عنصری بر می‌گرداند که

```
set1 = {'c', 'd', 'e', 'a'}
```

در دو یا چندین مجموعه مشترک هستند(اشتراک

```
result = mainSet.intersection(set1)
```

مجموعه‌ها در ریاضیات). توجه داشته باشید که آن

```
print(result)
```

مقادیر را در یک مجموعه جداگانه قرار می‌دهد.

```
# {'a', 'c'}
```

`intersection(set1,`

`set2, set3, ...)`

```
mainSet = {'a', 'b', 'c'}
```

نقش `=&` را بازی می‌کند و عنصری بر می‌گرداند

```
set1 = {'c', 'd', 'e', 'a'}
```

که در دو یا چندین مجموعه مشترک هستند. توجه

```
mainSet.intersection(set1)
```

داشته باشید که آن مقادیر را در یک مجموعه اولیه

```
print(mainSet)
```

قرار می‌دهد.

```
# {'a', 'c'}
```

`intersection_update(se`

`t1, set2, set3, ...)`

یک یا چندین مجموعه می‌گیرد.

```

mainSet = {'a', 'b', 'c'}           مقدار True یا False برمی‌گرداند و بررسی
set1 = {'c', 'd', 'e'}             می‌کند که آیا دو مجموعه عناصر مشترکی باهم
result =                           دارند یا نه. اگر حداقل یک عنصر مشترک داشته
mainSet.isdisjoint(set1)          باشند، مقدار True برمی‌گرداند.
print(result)                      یک مقدار می‌گیرد.

# True

```

```

mainSet = {'a', 'b', 'c'}           مقدار True یا False برمی‌گرداند و بررسی
set1 = {'c', 'a', 'd', 'e',       می‌کند که آیا تمامی عناصر مجموعه اول(اصلی) در
'b'}                            داخل مجموعه دوم موجود می‌باشد یا خیر.
result =                           یک مقدار می‌گیرد و آن هم مجموعه دوم.

mainSet.issubset(set1)            issubset(set2)
print(result)                      # True

```

```

mainSet = {'c', 'a', 'd',         مقدار True یا False برمی‌گرداند و بررسی
'e', 'b'}                         می‌کند که آیا تمامی عناصر مجموعه دوم در داخل
set1 = {'a', 'b', 'f'}             مجموعه اول(اصلی) موجود می‌باشد یا خیر.
result =                           یک مقدار می‌گیرد و آن هم مجموعه دوم.

mainSet.issuperset(set1)          issuperset(set2)
print(result)                      # False

```

```

Set1 = {'apple', 'orange',
        'cherry'}
Set2 = {'samsung',
        'xiaomi', 'apple'}
Result =
Set1.symmetric_difference
e(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
e_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 e_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’}

برای ادغام دو یا چند مجموعه استفاده می‌شود.

Set1. update(Set2)

نتیجه را در مجموعه اصلی جایگزین می کند.

`update(itrable)`

```
print(Set1)
```

لک با چند مقدار، (مجموعه) می‌گیرد.

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

۴-۶-۲) دسترسی به عناصر مجموعه

با توجه به نکات قبلی، امکان دسترسی مستقیم به عناصر مجموعه‌ها با آندیس 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 لیست اطمینان حاصل کرد. در ادامه برای درک این موضوع مثالی آورده شده است.

The screenshot shows a Python code editor interface with a sidebar of icons. The main area displays the following code:

```
sets.py
46
47 # Is Exist or NOT ???
48 set_example_11 = {"Reza", False, 23, 188.8, "Yellow", "1"}
49
50 print(f"Is 'Blue' in set_example_11: {'Blue' in set_example_11}")
51
52 print(f"Is '1' in set_example_11: {'1' in set_example_11}")
53
54 print(f"Is 'True' in set_example_11: {True in set_example_11}")
55
56 print(f"Is 'Reza' in set_example_11: {'Reza' in set_example_11}")
57
58 print(f"Is 23 in set_example_11: {23 in set_example_11}")
59
60
61
```

The terminal window at the bottom shows the execution of the script and its output:

```
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
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
PS C:\Users\RezaS\Desktop\python example>
```

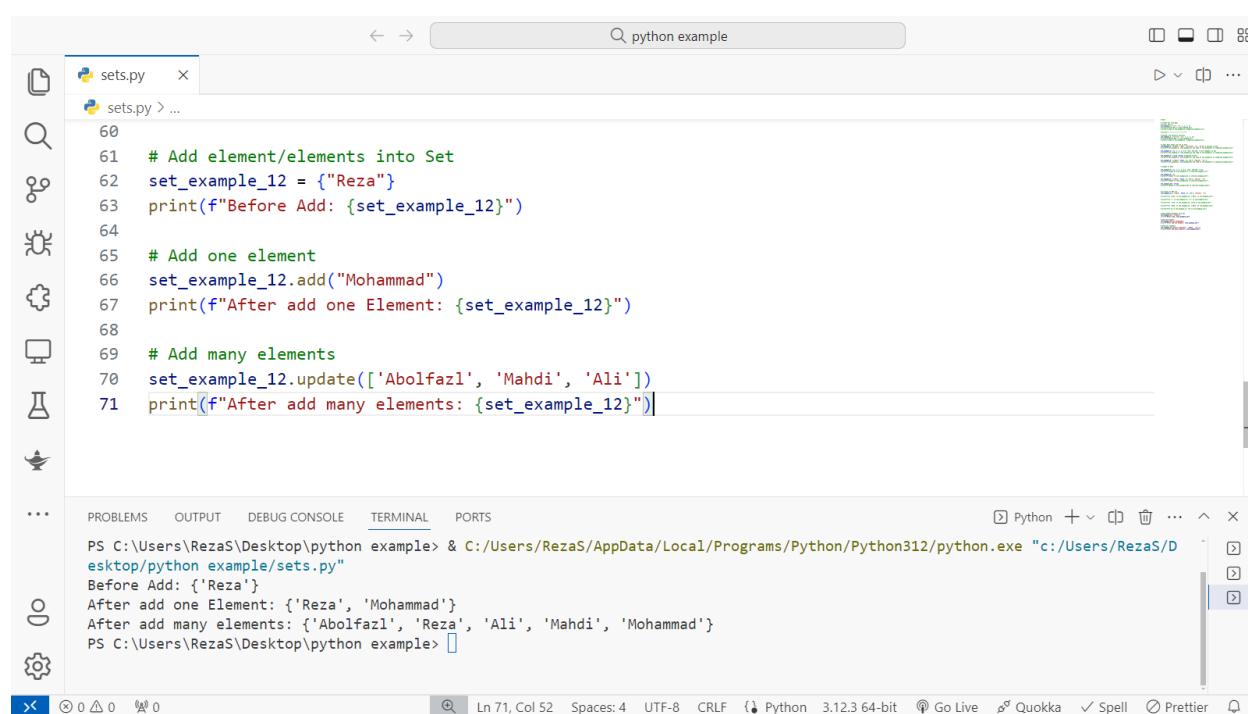
بررسی موجود بودن یک عنصر داخل یک مجموعه

۵-۶-۲-۴) افزودن عنصر/عناصر جدید به مجموعه‌ها

برای افزودن فقط یک عنصر به یک مجموعه، می‌توان از تابع وابسته `add` استفاده کرد. در ادامه به مثال زیر توجه کنید.

برای افزودن چند عنصر به یک سمت می‌توان از تابع وابسته `update` استفاده کرد.

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



```
sets.py
60
61 # Add element/elements into Set
62 set_example_12 = {"Reza"}
63 print(f"Before Add: {set_example_12}")
64
65 # Add one element
66 set_example_12.add("Mohammad")
67 print(f"After add one Element: {set_example_12}")
68
69 # Add many elements
70 set_example_12.update(['Abolfazl', 'Mahdi', 'Ali'])
71 print(f"After add many elements: {set_example_12}")
```

The screenshot shows a code editor window with a tab labeled "sets.py". The code itself is as follows:

```
sets.py
60
61 # Add element/elements into Set
62 set_example_12 = {"Reza"}
63 print(f"Before Add: {set_example_12}")
64
65 # Add one element
66 set_example_12.add("Mohammad")
67 print(f"After add one Element: {set_example_12}")
68
69 # Add many elements
70 set_example_12.update(['Abolfazl', 'Mahdi', 'Ali'])
71 print(f"After add many elements: {set_example_12}")
```

The terminal below the code shows the execution of the script:

```
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
Before Add: {'Reza'}
After add one Element: {'Reza', 'Mohammad'}
After add many elements: {'Abolfazl', 'Reza', 'Ali', 'Mahdi', 'Mohammad'}
PS C:\Users\RezaS\Desktop\python example>
```

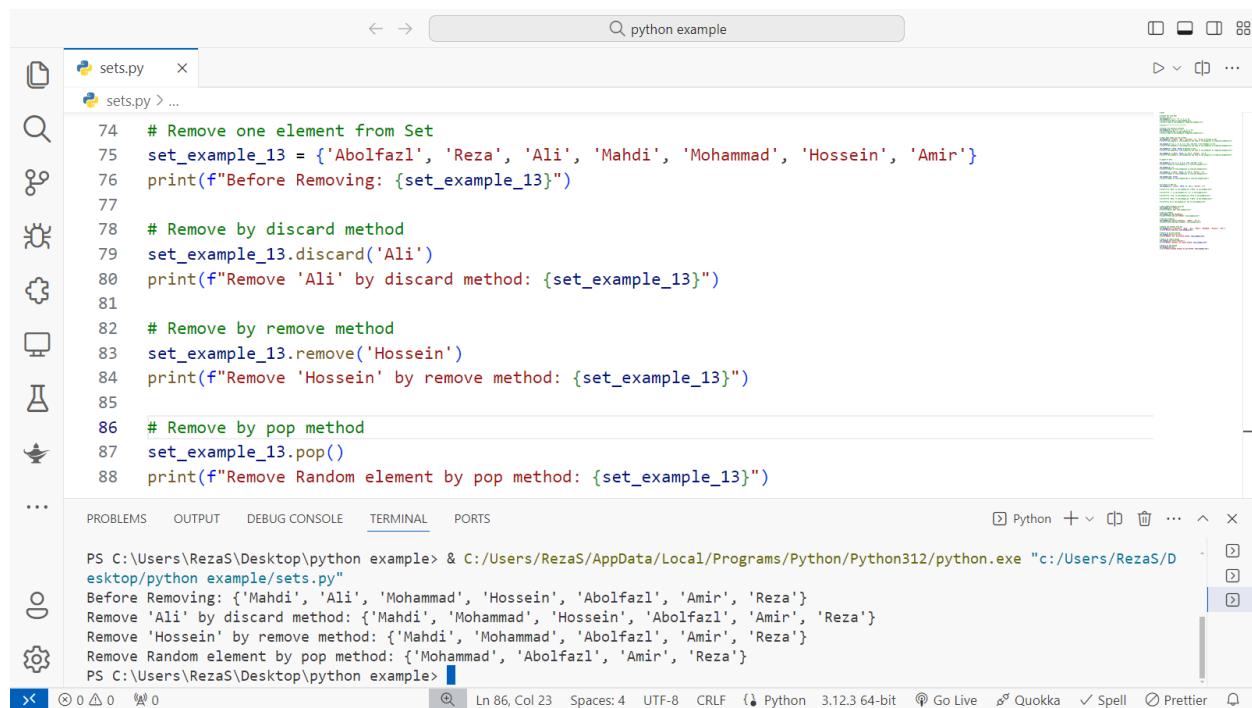
افزودن یک یا چند عنصر به داخل یک مجموعه

۴-۶-۲-۴) حذف عنصر از مجموعه یا حذف تمامی عناصر از مجموعه

معمولًا برای حذف یک عنصر مشخص از توابع وابسته‌ی `remove` و `discard` 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 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.



```
# Remove one element from Set
set_example_13 = {'Abolfazl', 'Reza', 'Ali', 'Mahdi', 'Mohammad', 'Hossein', 'Amir'}
print(f"Before Removing: {set_example_13}")

# Remove by discard method
set_example_13.discard('Ali')
print(f"Remove 'Ali' by discard method: {set_example_13}")

# Remove by remove method
set_example_13.remove('Hossein')
print(f"Remove 'Hossein' by remove method: {set_example_13}")

# Remove by pop method
set_example_13.pop()
print(f"Remove Random element by pop method: {set_example_13}")
```

The terminal output shows:

```
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
Before Removing: {'Mahdi', 'Ali', 'Mohammad', 'Hossein', 'Abolfazl', 'Amir', 'Reza'}
Remove 'Ali' by discard method: {'Mahdi', 'Mohammad', 'Hossein', 'Abolfazl', 'Amir', 'Reza'}
Remove 'Hossein' by remove method: {'Mahdi', 'Mohammad', 'Abolfazl', 'Amir', 'Reza'}
Remove Random element by pop method: {'Mohammad', 'Abolfazl', 'Amir', 'Reza'}
PS C:\Users\RezaS\Desktop\python example>
```

حذف یک عنصر از مجموعه با استفاده از توابع وابسته‌ی `remove` و `discard` و `pop`

معمولًا برای حذف تمامی عناصر یک مجموعه از تابع وابسته `clear` استفاده می‌شود. و نتیجه، یک مجموعه خالی و فاقد عنصر خواهد بود.

```

sets.py
sets.py > ...
90
91
92 # Remove all elements from Set
93
94 # remove by clear method
95 set_example_14 = {'Abolfazl', 'Reza', 'Ali', 'Mahdi', 'Mohammad', 'Hossein', 'Amir'}
96 print(f"Before Removing: {set_example_14}")
97
98 print("----- Cleaning -----")
99
100 set_example_14.clear()
101 print(f"After Removing: {set_example_14}")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
Before Removing: {'Hossein', 'Reza', 'Mahdi', 'Abolfazl', 'Amir', 'Mohammad', 'Ali'}
----- Cleaning -----
After Removing: set()
PS C:\Users\RezaS\Desktop\python example>

```

حذف تمامی عناصر مجموعه با استفاده از تابع وابسته `clear`

کلید واژه `del` نیز تمامی مجموعه را پاک خواهد کرد اما زمانی که شما آن نام آن مجموعه را فرا خوانید، خطای ارور خواهد داد.

۷-۶-۲-۴) ادغام دو یا چند مجموعه

برای ادغام دو یا چند مجموعه روش‌های مختلفی وجود دارد:

- توابع وابسته `update` و `union`
- تابع وابسته `intersection` (فقط مقادیر تکراری را برمی‌گرداند).
- تابع وابسته `difference` (مقادیری که در مجموعه اول بود و در مجموعه‌های دیگر نبوده را برمی‌گرداند).
- تابع وابسته `symmetric_difference` (همه مقادیر را برمی‌گرداند به جز مقادیر تکراری).

A screenshot of a Python code editor interface. The left sidebar shows icons for file, search, and other tools. The main area displays a Python script named 'sets.py' with the following code:

```
104 # union and update methods
105 set_example_15 = {'Abolfazl', 'Reza'}
106 adding_set_1 = set('Ali', 'Mahdi', 'Mohammad'))
107 adding_set_2 = {'Hossein', 'Amir'}
108 print(f"Before adding: {set_example_15}")
109
110 # union Method
111 myNewSet = set_example_15.union(adding_set_1)
112 print(f"adding by union method: {myNewSet}")
113
114 # update method
115 set_example_15.update(adding_set_2)
116 print(f"adding by update method: {set_example_15}")
```

The terminal tab at the bottom shows the output of running the script:

```
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
Before adding: {'Abolfazl', 'Reza'}
adding by union method: {'Mohammad', 'Abolfazl', 'Ali', 'Reza', 'Mahdi'}
adding by update method: {'Abolfazl', 'Reza', 'Hossein', 'Amir'}
PS C:\Users\RezaS\Desktop\python example>
```

استفاده از توابع وابسته‌ی union و update

A screenshot of a Python code editor interface. The left sidebar shows icons for file, search, and other tools. The main area displays a Python script named 'sets.py' with the following code:

```
118 # intersection and difference and symmetric_difference methods
119 set_example_16 = {'Abolfazl', 'Reza', 'Ali', 'Mahdi', 'Mohammad'}
120 set_example_17 = {'Hossein', 'Amir', 'Mahdi', 'Reza', 'Ali'}
121
122 # intersection method
123 intersection_result = set_example_16.intersection(set_example_17)
124 print(f"intersection of set_example_16 and set_example_17 is {intersection_result}")
125
126 # difference method
127 difference_result = set_example_16.difference(set_example_17)
128 print(f"difference of set_example_16 and set_example_17 is {difference_result}")
129
130 # symmetric_difference methods
131 symmetric_difference_result = set_example_16.symmetric_difference(set_example_17)
132 print(f"symmetric_difference of set_example_16 and set_example_17 is {symmetric_difference_result}")
133
```

The terminal tab at the bottom shows the output of running the script:

```
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
intersection of set_example_16 and set_example_17 is {'Mahdi', 'Reza', 'Ali'}
difference of set_example_16 and set_example_17 is {'Abolfazl', 'Mohammad'}
symmetric_difference of set_example_16 and set_example_17 is {'Amir', 'Hossein', 'Mohammad', 'Abolfazl'}
PS C:\Users\RezaS\Desktop\python example>
```

استفاده از توابع وابسته‌ی symmetric_difference و difference .intersection

نکات:

✓ می‌توانید به جای تابع وابسته `union` از علامت | برای ادغام دو یا چند مجموعه استفاده کنید. البته توجه

داشته باشید 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 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.

✓ می‌توانید به جای تابع وابسته `symmetric_difference` از علامت ^ بین دو یا چندین مجموعه استفاده

کنید. برای استفاده از این علامت نیز باید تمامی داده‌ها از نوع `set` باشند.

✓ مقادیر 1 و `True` یکسان هستند و همچنین مقادیر 0 و `False`. بنابراین زمانی که در یک مجموعه عناصر

1 و `True` وجود داشته باشد، 1 حذف خواهد شد چرا که با `True` یکسان می‌باشد. برای 0 و `False` نیز

این مورد صحت دارد.

The screenshot shows a Python code editor interface with a file named 'sets.py' open. The code demonstrates set operations:

```
134
135     set_example_18 = {'Abolfazl', 'Reza'}
136     set_example_19 = {'Mahdi', 'Reza', 'Ali'}
137     set_example_20 = {'Reza', 'Hossein', 'Amir'}
138     set_example_21 = {'Hossein', 'Reza', 'Ali'}
139
140     # union =
141     union_result = set_example_18 | set_example_19 | set_example_20 | set_example_21
142     print(f"Result of union is {union_result}")
143
144     # intersection =
145     intersection_result = set_example_18 & set_example_19 & set_example_20 & set_example_21
146     print(f"Result of intersection is {intersection_result}")
```

The terminal output shows the results of the operations:

```
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
Result of union is {'Hossein', 'Abolfazl', 'Mahdi', 'Reza', 'Amir', 'Ali'}
Result of intersection is {'Reza'}
```

نکاتی درباره توابع وابسته‌ی union و intersection

The screenshot shows a Python code editor interface with a file named 'sets.py' open. The code demonstrates set operations:

```
148     set_example_18 = {'Abolfazl', 'Reza'}
149     set_example_19 = {'Mahdi', 'Reza', 'Ali'}
150     set_example_20 = {'Reza', 'Hossein', 'Amir'}
151     set_example_21 = {'Hossein', 'Reza', 'Ali'}
152
153     # difference =
154     difference_result = set_example_18 - set_example_19 - set_example_20 - set_example_21
155     print(f"Result of difference is {difference_result}")
156
157     # symmetric_difference =
158     symmetric_difference_result = set_example_18 ^ set_example_19 ^ set_example_20 ^ set_example_21
159     print(f"Result of symmetric_difference is {symmetric_difference_result}")
```

The terminal output shows the results of the operations:

```
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
Result of difference is {'Abolfazl'}
Result of symmetric_difference is {'Amir', 'Mahdi', 'Abolfazl'}
```

نکاتی درباره توابع وابسته‌ی symmetric_difference و difference

The screenshot shows a code editor interface with a Python file named 'sets.py' open. The code demonstrates various ways to create sets in Python:

```
sets.py
161 # True = 1
162 set_example_22 = {True, 1}
163 print(f"Result: {set_example_22}")
164
165 # False = 0
166 set_example_23 = {False, 0}
167 print(f"Result: {set_example_23}")
168
169 # Combination
170 set_example_24 = {1, False, 0, True}
171 print(f"Result: {set_example_24}")
```

Below the code editor is a terminal window showing the execution of the script:

```
PS C:\Users\RezaS\Desktop\python example> & C:/Users/RezaS/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaS/Desktop/python example/sets.py"
Result: {True}
Result: {False}
Result: {False, 1}
PS C:\Users\RezaS\Desktop\python example>
```

نکاتی درباره مقادیر True و False در داخل مجموعه‌ها

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 به فردی دارد، در این صورت برای این که 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 خطای ریافت کند.

1-۵) 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 لیست‌ها، 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:

if شرط:

دستورات

اگر شرط نوشته شده در بخش if ارزیابی شود و نتیجه‌ی آن True باشد، کدهای داخل بلک if اجرا می‌شوند. در غیر این صورت، این دستورات نادیده گرفته می‌شوند.

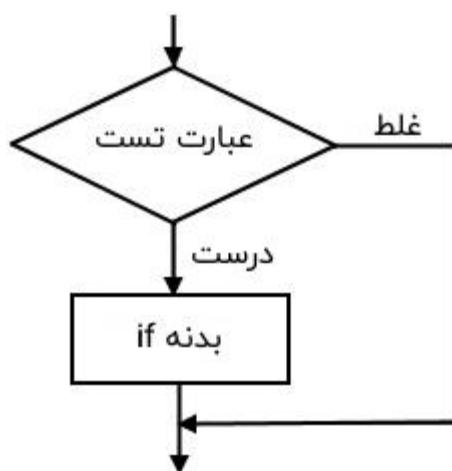
نکته: پایتون یک زبان حساس به دندانه‌گذاری^{۴۵} است و استفاده صحیح از آن برای ساختاردهی کد الزامی است.

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 یا ۱ تب).

فلوچارت مربوط به دستورات شرطی:



⁴⁵ Indentation

ساختار دستورات شرطی:

- ساختار اول: ساختاری است که فقط با `if` تشکیل شده است.
- 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^{۴۶}) می‌آید و سپس بعد از شرط، علامت دو نقطه روی هم (:) نوشته می‌شود. و سپس در خط بعدی ابتدا دندانه‌گذاری و سپس دستورالعمل‌ها^{۴۷} به صورت کد نوشته می‌شود.

اغلب زبان‌های برنامه‌نویسی مانند C,C++ 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

⁴⁶ Condition

⁴⁷ Statement

```

# Condition Statement
"""
if condition:
    statement
"""

x = 25
if x%2 == 1:
    print(f"x:{x} is Odd number")
print("-----")
name = "Reza"
if name == "ali":
    print("Hello Ali")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/New folder/Python/python example/condition_Statement.py"
x:25 is Odd number

PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>

مثالی از عبارات شرطی فقط با if

۲-۱-۵) ساختار if/else

قسمت اول این ساختار دقیقا همانند ساختار اول می‌باشد، اما در قسمت کلیدواژه else به ساختار اضافه می‌شود. به این صورت که ساختار اول را نوشته و دستورالعمل‌های if را نوشته و سپس در خط بعدی بدون دندانه‌گذاری، else را نوشته و سپس بعد از کلیدواژه else دو نقطه روی هم گذاشته می‌شود (برای کلیدواژه else شرط گذاشته نمی‌شود). در نهایت بعد از گذاشتن دونقطه روی هم، در خط بعدی ابتدا دندانه‌گذاری می‌شود و سپس دستورالعمل‌های مربوط به else نوشته می‌شود. منظور از else این است که اگر شرط در if رد شود، دستورات else را اجرا شود.

ساختار کلی:

if condition:

statement

else:

statement

```

condition_Statement.py > ...
21 # Second Syntax (if/else)
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 print(" ----- ")
36 user_score = 5
37 # if user score grater than 10, user pass else Reject
38 if user_score > 10:
39     print("Pass.")
40 else:
41     print("Reject.")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

1254 is Even number.

Reject.

PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>

Ln 20, Col 1 Spaces: 4 UTF-8 CRLF Python 3.12.4 64-bit

مثالی از عبارات شرطی با if/else

۳-۱-۵ if/elif/else ساختار

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 کلیدوازه `elif` شروع می‌شود. به این صورت نوشته می‌شود که بعد از دستورالعمل‌های `if`، در خط جدید، کلیدوازه `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 کلیدوازه `else` می‌باشد که همانند ساختار دوم نوشته می‌شود.

نکته: ممکن است یک برنامه دو شرط داشته باشد، 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 که در این صورت باید برای هر شرط یک `if` گذاشته شود.

ساختار کلی:

دو شرط:

`if condition_1:`

statement

`elif condition_2:`

statement

`else:`

statement

```
condition_Statement.py X
condition_Statement.py > ...
43
44 # Third Syntax (if/elif/else)
45 # Two Condition
46
47 user_score = 17
48
49 if user_score >= 17:
50     print("Good Job.")
51 elif 10 < user_score < 17:
52     print("Not Bad.")
53 else:
54     print("It's Bad.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python .exe "c:/Users/RezaSeyyed/Desktop/New folder/Python/python example/condition_Statement.py"
Good Job.
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>
```

مثالی از عبارات شرطی با if/elif/else

۴-۱-۵) ساختار if/elif/else با بیش از دو شرط:

If condition_1:

statement

elif condition_2:

statement

elif condition_3:

statement

...

else:

statement

```

condition_Statement.py
57 # Third Syntax (if/elif/else)
58 # Grater than Two Condition
59
60 user_color = "yellow"
61 if user_color == "green":
62     print("First try: User color is green.")
63 elif user_color == "red":
64     print("Second try: User color is red.")
65 elif user_color == "black":
66     print("Third try: User color is black.")
67 elif user_color == "white":
68     print("Fourth try: User color is white.")
69 elif user_color == "yellow":
70     print("Fifth try: User color is yellow.")
71 elif user_color == "blue":
72     print("Last try: User color is blue.")
73 else:
74     print("Oh no, I can't guess color! :(")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python .exe "c:/Users/RezaSeyyed/Desktop/New folder/Python/python example/condition_Statement.py"
Fifth try: User color is yellow.
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>

x 0 △ 0 ⌂ 0

Ln 67, Col 28 Spaces: 4 UTF-8 CRLF Python 3.12.4 64-bit

مثالی از عبارات شرطی با if/elif/else

شرطهایی که می‌توانید در دستورات شرطی استفاده کنید:

a == b b: برابر است با a •

a != b b: با a برابر نیست: •

a < b b: a کوچکتر است از •

a <= b b: a کوچکتر یا مساوی •

a > b b: a بزرگتر است از •

$a \geq b$ بزرگتر یا مساوی b •

```
# Conditions
a = 45
b = 13
if a == b:
    print("a and b are equal.")
if a != b:
    print("a and b are NOT equal.")
if a > b:
    print("a is grather than b.")
if a >= b:
    print("a is greater than or equal to b.")
if a < b:
    print("a is less than b.")
if a <= b:
    print("a is less than or equal to b.")

.exe "c:/Users/RezaSeyyed/Desktop/New folder/Python/python example/condition_Statement.py"
a and b are NOT equal.
a is grather than b.
a is greater than or equal to b.
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>
```

تعدادی از شرط‌ها که می‌توان در عبارات شرطی استفاده کرد

کلیدواژه‌هایی که بین دو شرط استفاده می‌شود:

• استفاده از کلیدواژه `or` بین دو شرط: اگر فقط نتیجه‌ی یکی از شرط‌ها مقدار `True` داشته باشد،

دستورالعمل‌ها اجرا خواهند شد.

• استفاده از کلیدواژه `and` بین دو شرط: اگر نتیجه هر دو شرط `True` باشند، دستورالعمل‌ها اجرا خواهند

شد.

```
# And/Or in Condition Statement
name = "Reza"
age = 23

if name == "Reza" and age == 24:
    print("By And")
    print(f"name = {name}:{name == 'Reza'}")
    print("AND")
    print(f"age = 24: {age == 24}")
    print(f"Result is {name == 'Reza' and age == 24}")

if name == "Reza" or age == 24:
    print("By Or")
    print(f"name = {name}:{name == 'Reza'}")
    print("OR")
    print(f"age = 24: {age == 24}")
    print(f"Result is {name == 'Reza' or age == 24}")

By Or
name = 'Reza':True
OR
age = 24: False
Result is True
```

مثالی از استفاده `and` و `or` در عبارات شرطی

نکته: استفاده از کلیدواژه `in` برای بررسی وجود داشتن یا نداشتن یک عنصر در داخل لیست، تاپل و هر نوع داده‌ای که قابلیت پیمایش دارد.

The screenshot shows a Python code editor interface with the following details:

- File Explorer:** Shows a file named "condition_Statement.py".
- Code Editor:** Displays the following Python code:

```
119
120     # in keyword in condition statement
121 colors_list = ["red", "green", "blue", "black", "white"]
122
123 if "yellow" in colors_list:
124     print("yellow is exist.")
125 else:
126     print("yellow is NOT exist.")
127
128 print("-----")
129
130 if 'blue' in colors_list:
131     print("blue is exist.")
132 else:
133     print("blue is NOT exist.")
134
```
- Terminal:** Shows the command-line output of running the script:

```
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python .exe "c:/Users/RezaSeyyed/Desktop/New folder/Python/python example/condition_Statement.py"
yellow is NOT exist.
-----
blue is exist.
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>
```

مثالی از کلیدواژه `in` داخل عبارات شرطی

استفاده از کلیدواژه **not** در جملات شرطی

معمول از این کلیدواژه جهت ضد کردن یک شرط استفاده می‌شود. برای مثال اگر عنصری در داخل لیستی وجود نداشته باشد، دستور `العمل`‌های زیر را اجرا شود.

```

condition_Statement.py X
condition_Statement.py > ...

136 # not keyword in condition statement
137 colors_list = ["red", "green", "yellow", "orange", "white"]
138
139 if "yellow" not in colors_list:
140     print("yellow is NOT exist.")
141 else:
142     print("yellow is exist.")
143
144 print("-----")
145
146 if 'blue' not in colors_list:
147     print("blue is NOT exist.")
148 else:
149     print("blue is exist.")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + v 🌐 ... ^ x
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/New folder/Python/python example/condition_Statement.py"
yellow is exist.
-----
blue is NOT exist.
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>

```

x 0 ▲ 0 🔍 0

Ln 145, Col 1 Spaces: 4 UTF-8 CRLF { Python 3.12.4 64-bit 🔍

مثالی از کلیدواژه `not` در عبارات شرطی

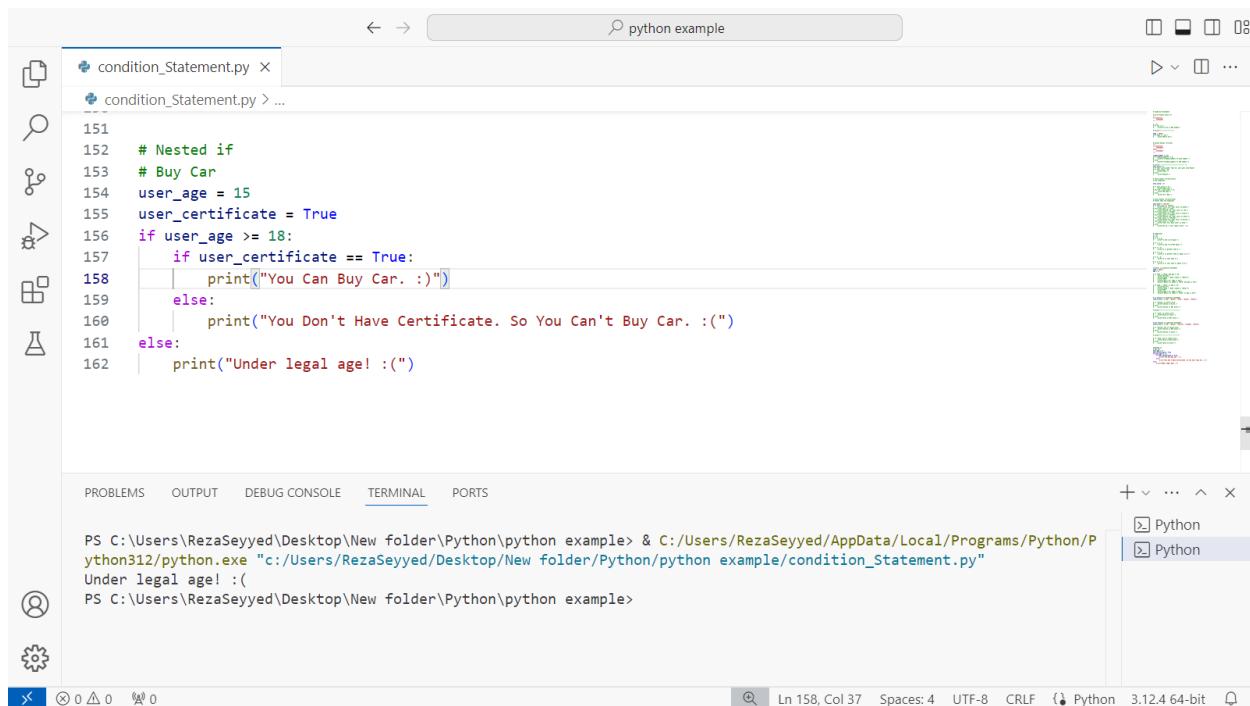
۴-۱-۵) دستورات شرطی تو در تو^{۴۸}

یکی از قابلیت‌هایی که زبان برنامه‌نویسی پایتون دارد، استفاده از دستورات شرطی داخل دستورات شرطی دیگر است. برای مثال شما می‌توانید از دستورات شرطی که در مطالب پیشیت توضیح داده شد، در داخل دستورات شرطی قرار دهید.

برای درک بهتر این موضوع یک مثالی از خرید خوردو 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، دارا بودن یا نبودن گواهینامه رانندگی کاربر می‌باشد، اگر کاربر بیشتر از هجده سال داشته باشد و

⁴⁸ Nested if Statement

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.



```
# Nested if
# Buy Car
user_age = 15
user_certificate = True
if user_age >= 18:
    if user_certificate == True:
        print("You Can Buy Car. :)")
    else:
        print("You Don't Have Certificate. So You Can't Buy Car. :(")
else:
    print("Under legal age! :(")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/New folder/Python/python example/condition_Statement.py"
Under legal age! :(
```

PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>

مثالی از عبارات شرطی تو در تو

```
151
152 # Nested if
153 # Buy Car
154 user_age = 24
155 user_certificate = False
156 if user_age >= 18:
157     if user_certificate == True:
158         print("You Can Buy Car. :)")
159     else:
160         print("You Don't Have Certificate. So You Can't Buy Car. :(")
161 else:
162     print("Under legal age! :(")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/New folder/Python/python example/condition_Statement.py"
You Don't Have Certificate. So You Can't Buy Car. :(
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>
```

مثالی از عبارات شرطی تو در تو

```
151
152 # Nested if
153 # Buy Car
154 user_age = 24
155 user_certificate = True
156 if user_age >= 18:
157     if user_certificate == True:
158         print("You Can Buy Car. :)")
159     else:
160         print("You Don't Have Certificate. So You Can't Buy Car. :(")
161 else:
162     print("Under legal age! :(")

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/New folder/Python/python example/condition_Statement.py"
You Can Buy Car. :(
PS C:\Users\RezaSeyyed\Desktop\New folder\Python\python example>
```

مثالی از عبارات شرطی تو در تو

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 این‌که 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

حلقه‌ها^{۴۹} 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.

۱-۶) 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، بدنی 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) به طور خاص مناسب ساخته است.

ساختار کلی حلقه `while`

⁴⁹ Loops

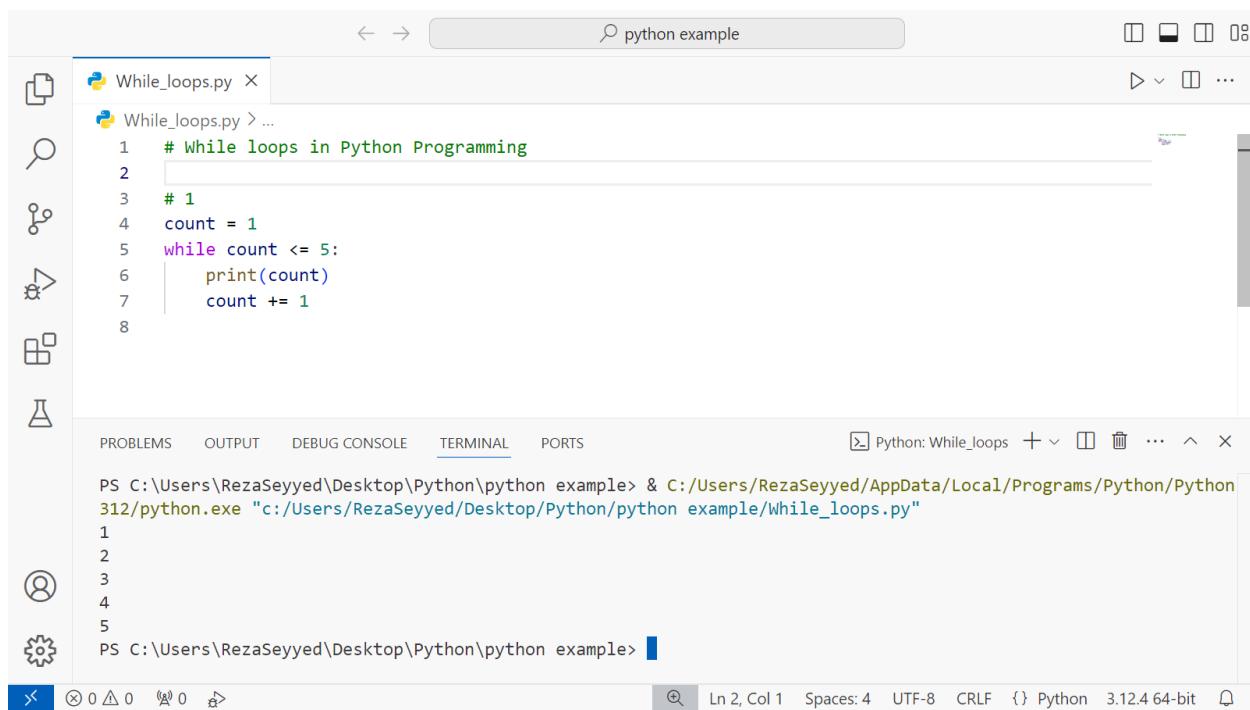
While condition:

statement

نکته: دندانه‌گذاری در بدن حلقه (بخش دستورالعمل‌ها) ضروری است.

۱-۱-۶) مثال‌های مربوط به حلقه **while**

مثال ۱. با استفاده از حلقه **while** برنامه بنویسید که اعداد ۱ تا ۵ چاپ شود.



The screenshot shows the Visual Studio Code interface. On the left is the sidebar with icons for files, search, symbols, and others. The main area has a tab for 'While_loops.py'. The code editor contains the following Python script:

```
1 # While loops in Python Programming
2
3 # 1
4 count = 1
5 while count <= 5:
6     print(count)
7     count += 1
8
```

Below the code editor is the terminal window, which shows the output of running the script:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/While_loops.py"
1
2
3
4
5
```

The status bar at the bottom indicates the file is saved (green dot), the line and column are Ln 2, Col 1, and the Python version is 3.12.4 64-bit.

این کد نمونه‌ای ساده از استفاده از حلقه‌ی `while` در پایتون است که اعداد از ۱ تا ۵ را چاپ می‌کند.

۱. تعریف متغیر: ابتدا متغیر `count` با مقدار ۱ تعریف می‌شود.

۲. شروع حلقه و برقرار بودن شرط حلقه: حلقه‌ی `while` بررسی می‌کند که آیا مقدار `count` کمتر یا مساوی

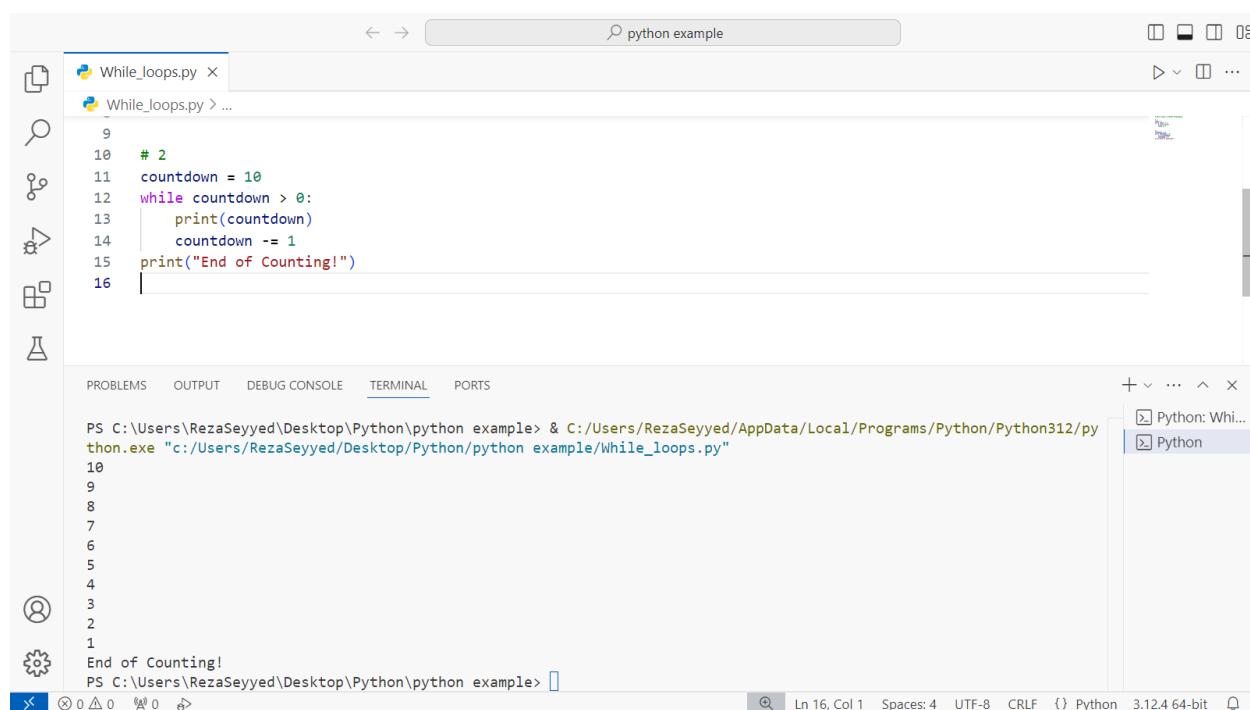
۵ است. اگر شرط برقرار باشد، حلقه اجرا می‌شود.

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.

مثال ۲. با استفاده از حلقه while برنامه شمارش معکوس از ۱۰ به ۱ بنویسید.



```
# 2
countdown = 10
while countdown > 0:
    print(countdown)
    countdown -= 1
print("End of Counting!")
```

این کد یک شمارش معکوس از ۱۰ تا ۱ را نمایش می‌دهد و در پایان عبارت "پایان شمارش!" را چاپ می‌کند.

در اینجا شرح مراحل اجرای کد آورده شده است:

۱. تعریف متغیر: متغیر `countdown` با مقدار اولیه ۱۰ تعریف شده است.

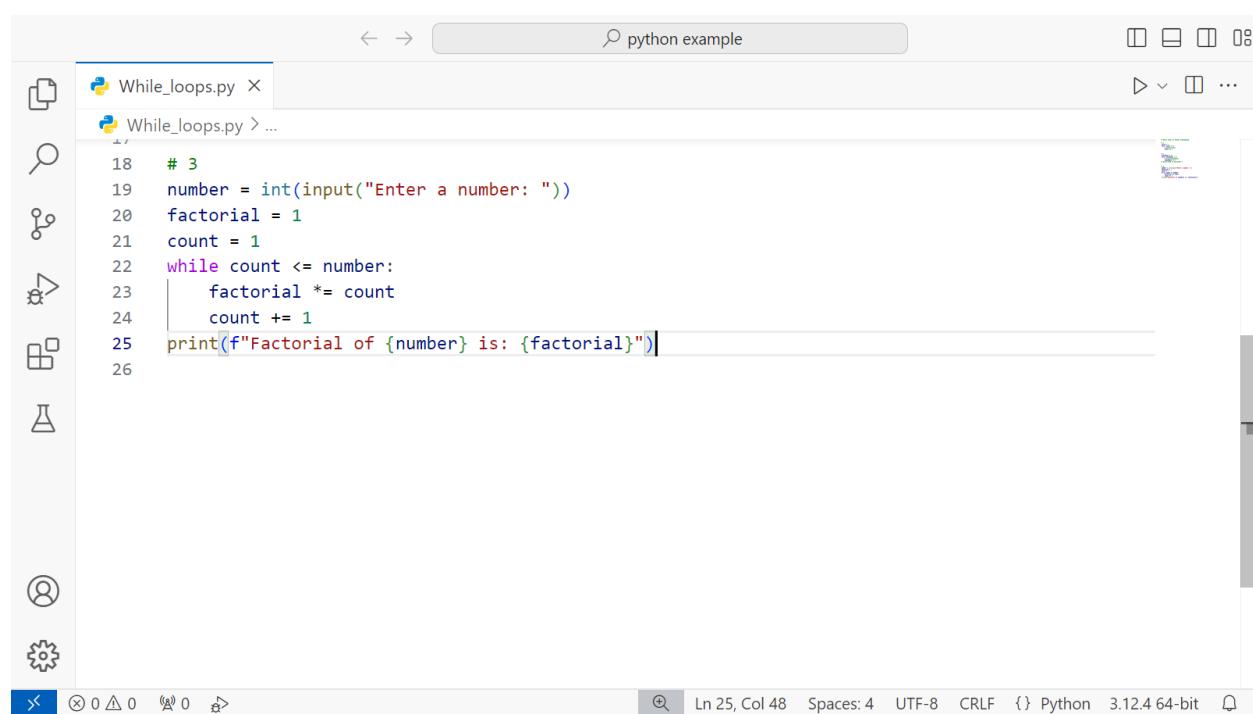
۲. شروع حلقه‌ی `while` و برقراری شرط حلقه: حلقه‌ی `while` برسی می‌کند که آیا مقدار `countdown` بزرگتر از صفر است. اگر این شرط برقرار باشد، حلقه اجرا می‌شود.

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 `countdown > 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
While_loops.py
While_loops.py > ...
18 # 3
19 number = int(input("Enter a number: "))
20 factorial = 1
21 count = 1
22 while count <= number:
23     factorial *= count
24     count += 1
25 print(f"Factorial of {number} is: {factorial}")
26
```

این کد به منظور محاسبه‌ی فاکتوریل یک عدد استفاده می‌شود. در ادامه توضیح دقیق مراحل اجرای آن آورده

شده است:

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: *****

```
python example
While_loops.py
While_loops.py > ...
18 # 3
19 number = int(input("Enter a number: "))
20 factorial = 1
21 count = 1
22 while count <= number:
23     factorial *= count
24     count += 1
25 print(f"Factorial of {number} is: {factorial}")
26
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/While_loops.py"
Enter a number: 5
Factorial of 5 is: 120
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
+ ... ^ x
Python: Whi...
Python
Ln 25, Col 48 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit
```

۲-۱-۶ while در حلقه else بخش

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، بلکه 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 می‌خواهیم در یک حلقه، عددی را تا زمانی که کمتر از ۵ است افزایش دهیم. اگر این عدد به ۵ یا بیشتر برسد، حلقه به طور طبیعی خاتمه می‌یابد و بخش `else` اجرا می‌شود. اما اگر به دلیل شرایط خاصی از حلقه خارج شویم (مانند استفاده از `break`)، بخش `else` نادیده گرفته خواهد شد. این ویژگی به ما این امکان را می‌دهد تا بین خاتمه‌ی طبیعی و خاتمه‌ی غیرمنتظره‌ی حلقه تمایز قائل شویم.

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.

```

28 # ** Else Statement in While loop **
29
30 count = 1
31 while count <= 5:
32     print(count)
33     count += 1
34 else:
35     print("End of the Loop!")
36
37

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/While_loops.py"
1
2
3
4
5
End of the Loop!
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

```

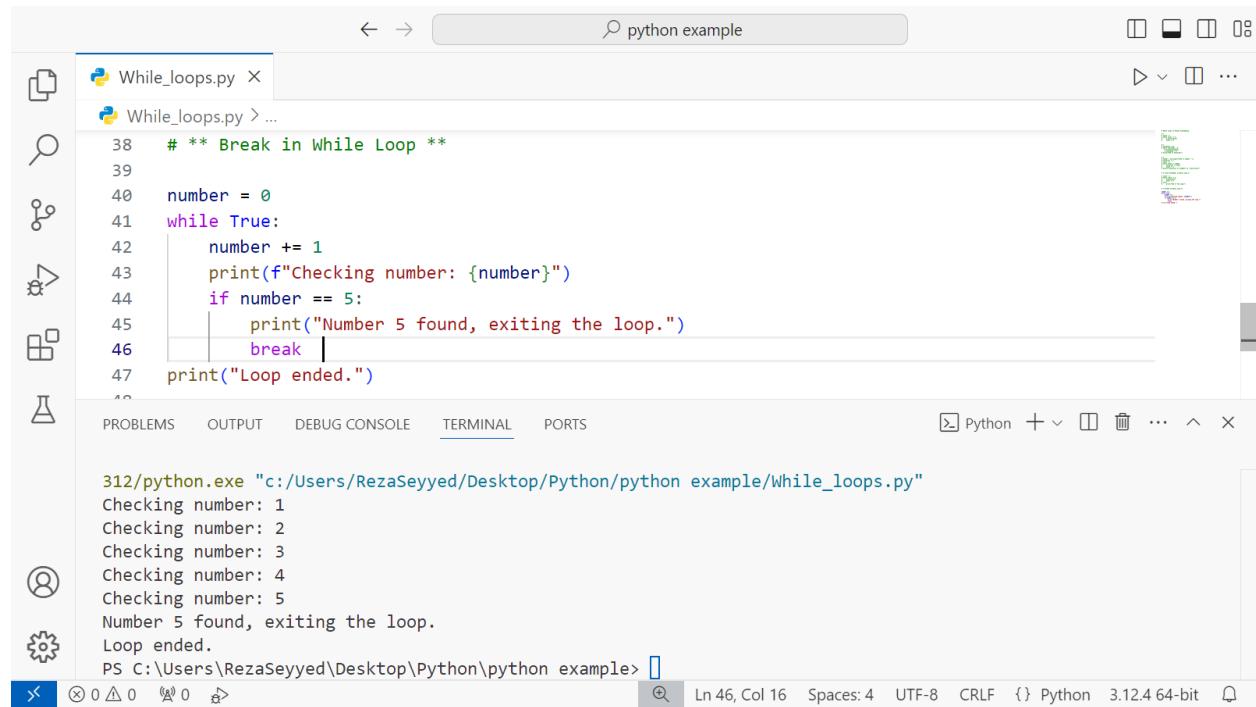
۳-۱-۶ دستور `while` در حلقه و `break` و `continue`

در زبان برنامه‌نویسی پایتون، دستورهای `break` و `continue` بعنوان دو ابزار کلیدی برای کنترل جریان اجرای حلقه‌ها، بهویژه 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؛ به‌طوری‌که 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 اجبار 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 می‌خواهیم عددی را بررسی کنیم و به محض پیدا کردن یک مقدار خاص، از حلقه خارج شویم. در این حالت، دستور `break` به ما اجازه می‌دهد که با رسیدن به مقدار مورد نظر، حلقه را خاتمه دهیم.



```

38 # ** Break in While Loop **
39
40     number = 0
41     while True:
42         number += 1
43         print(f"Checking number: {number}")
44         if number == 5:
45             print("Number 5 found, exiting the loop.")
46             break
47     print("Loop ended.")

```

312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/While_loops.py"
 Checking number: 1
 Checking number: 2
 Checking number: 3
 Checking number: 4
 Checking number: 5
 Number 5 found, exiting the loop.
 Loop ended.

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.

۱-۳-۲) دستور `continue`

دستور `continue`، برخلاف `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 بخواهیم در یک حلقه‌ی `while` تنها مقادیری را که مضربی از ۳ نیستند بررسی کنیم، می‌توانیم از `continue` استفاده کنیم تا به محض برخورد با مضرب ۳، آن مرحله خاص را رد کرده و به مرحله بعدی برویم.

```

50  # ** Continue in While Loop **
51
52  number = 0
53  while number < 10:
54      number += 1
55      if number % 3 == 0:
56          continue
57      print(f"Checking number: {number}")
58

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```

312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/While_loops.py"
Checking number: 1
Checking number: 2
Checking number: 4
Checking number: 5
Checking number: 7
Checking number: 8
Checking number: 10
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

```

Ln 58, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit

در این مثال:

- 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 لیست‌ها، مجموعه‌ها، رشته‌ها، یا حتی یک دامنه از اعداد باشد (نوع داده‌های که قابلیت پیمایش دارند). بر خلاف حلقه‌ی `while` که مبنی 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. این شیوه از تکرار، فرآیند کار با مجموعه‌ها و توالی‌های داده را ساده و قابل فهم می‌کند.

ساختار حلقه‌ی `for` در پایتون به این شکل است که با استفاده از عبارت `for item 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 بدنه حلقه (بخش دستورالعمل‌ها) ضروری است.

برای درک بهتر، فرض کنید می‌خواهیم اعداد ۱ تا ۵ را در یک لیست چاپ کنیم. به‌کمک حلقه‌ی `for` می‌توانیم به‌سادگی به هر عدد دسترسی پیدا کرده و آن را نمایش دهیم. به عنوان مثال:

```
# For Loops in Python
#
# 1
for number in [1,2,3,4,5]:
    print(number)

```

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/For_loops.py"
1
2
3
4
5
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

در این مثال، در هر تکرار مقدار جدیدی از لیست 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:

```
For_loops.py > ...
7
8 # Range in loops
9
10 for i in range(5):
11     print(f"This number is: {i}")
12

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/For_loops.py"
This number is: 0
This number is: 1
This number is: 2
This number is: 3
This number is: 4
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

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 لیست می‌توان از حلقه‌ی for برای بررسی

تک تک 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

مثال زیر، این مفهوم را به خوبی روشن می‌سازد:

```

13
14  # ** Else Statement in For loop **
15
16  items = [2, 4, 6, 8]
17  target = 5
18  for item in items:
19      if item == target:
20          print("Not Found the Number!:(")
21          break
22  else:
23      print("list index out of range!:( ")

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/For_loops.py"
list index out of range!:(
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

Ln 22, Col 6 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit

در این مثال، حلقه تمام عناصر لیست 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 لیست 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:

```

25
26 # Example 1: Student's Score
27
28 grades = [18, 20, 17, 19, 15] # لیست نمرات دانشجو
29
30 total = 0 # متغیر برای جمع نمرات
31
32 for grade in grades: # حلقه برای جمع کردن نمرات
33     total += grade
34
35 average = total / len(grades) # محاسبه میانگین
36
37 print("Averge of Scores: ", average) # نمایش میانگین
38

```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/For_loops.py"
Averge of Scores: 17.8
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

Ln 38, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit

در این کد:

۱. ابتدا لیستی به نام `grades` برای ذخیره‌ی نمرات دانشجو تعریف شده است.
۲. سپس یک متغیر به نام `total` برای جمع نمرات ایجاد کرده‌ایم و مقدار اولیه‌ی آن را برابر با صفر قرار داده‌ایم.
۳. با استفاده از حلقه‌ی `for`، تک‌تک نمرات را از لیست `grades` دریافت کرده و به `total` اضافه می‌کنیم.
۴. 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.

```

For_loops.py x
For_loops.py > [e] main_numbers_list
40 # Example 2: Separate Even and Odd numbers from a list
41 main_numbers_list = [45, 2, 49, 17, 65, 98, 11, 83, 15, 32]
42
43 even_numbers_list = [] # Even numbers list
44 odd_numbers_list = [] # Odd numbers list
45
46 for i in main_numbers_list:
47     if i%2 == 0: # If the number is even
48         even_numbers_list.append(i)
49     elif i%2 != 0:
50         odd_numbers_list.append(i)
51     else:
52         print("The number is not even or odd!")
53
54 # Print the results
55 print(f"The Even numbers list is: {even_numbers_list}")
56 print(f"The Odd numbers list is: {odd_numbers_list}")

```

فهرست اعداد تصادفی #

ایجاد فهرست برای افزودن اعداد زوج
ایجاد فهرست برای افزودن اعداد فرد

استفاده از حلقه برای پیمایش بر روی فهرست اعداد تصادفی # استفاده از دستورات شرطی برای یافتن اعداد زوج و فرد و جداسازی آن ها

چاپ کردن فهرست اعداد زوج و فرد به صورت جداگانه

The Even numbers list is: [2, 98, 32]
The Odd numbers list is: [45, 49, 17, 65, 11, 83, 15]

در این کد:

۱. ابتدا یک فهرست از اعداد تصادفی ایجاد کرده و سپس دو فهرست برای اعداد زوج و فرد به صورت جداگانه ایجاد می‌کنیم.

۲. 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.

۳. 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.

The screenshot shows the Visual Studio Code interface with the title bar "python example". In the left sidebar, there are icons for file operations, a search bar, and tabs for "For_loops.py" and "strings.py". The main editor area contains Python code for separating characters from a string:

```
59 # Example 3: Separate Character of String.  
60  
61 string_example = "Apples"  
62 char_list = []  
63  
64 for char in string_example:  
65     char_list.append(char)  
66     print(f"The character in this loop is: {char}")  
67  
68 print(f"List of Characters: {char_list}")
```

The terminal below shows the execution of the script and its output:

```
312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/For_loops.py"  
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']  
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

در این کد:

۱. ابتدا یک رشته‌ی تصادفی مانند **Apples** را ایجاد کرده و درون متغیر **string_example** قرار می‌دهیم.
۲. همچنین یک فهرست برای افزودن کاراکترهای رشته ایجاد می‌کنیم.
۳. برای پیمایش بر روی رشته ایجاد شده باید از حلقه **for** استفاده کنیم. در داخل حلقه **for**، در هر تکرار، کاراکتر مورد نظر آن رشته ابتدا به فهرست ایجاد شده اضافه می‌شود و سپس آن کاراکتر در خروجی چاپ می‌شود.
۴. 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.

```

# Example 4: Separate keys and values from a Dictionary.
main_dict = {
    "name": "Reza",
    "age": 23,
    "job": "Student",
    "mobile": "09011111111",
    "favorite color": "yellow"
}
keys_list = []
values_list = []

for key, value in main_dict.items():
    keys_list.append(key)
    values_list.append(value)

print(f"The Keys of main_dict are: {keys_list}")
print(f"The Values of main_dict are: {values_list}")

```

The Keys of main_dict are: ['name', 'age', 'job', 'mobile', 'favorite color']
The Values of main_dict are: ['Reza', 23, 'Student', '09011111111', 'yellow']

در این کد:

۱. ابتدا یک دیکشنری تصادفی از کلید و مقدارهای تصادفی ایجاد می‌کنیم.
۲. سپس دو فهرست برای افزودن کلید و مقدار دیکشنری به صورت جداگانه ایجاد می‌کنیم.
۳. با استفاده از حلقه **for** و با استفاده از تابع وابسته **items**، به دیکشنری اختصاص می‌دهیم و در داخل حلقه، کلیدها و مقدارهای موجود در هر تکرار را به فهرست های ایجاد شده، اضافه می‌کنیم.
۴. در نهایت فهرست های 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 موردنظر شامل حاصل ضرب همه اعداد خواهد بود. در ادامه، کد مربوطه را مشاهده می‌کنید:

The screenshot shows a code editor window titled "python example". The file "For_loops.py" contains the following code:

```
89
90  # Example 5: multiple of the numbers list
91
92  numbers_list = [2, 7, 1, 3] # فهرست نمونه‌ای از اعداد
93  multiple = 1 # متغیر برای ذخیره حاصل‌ضرب، مقدار اولیه ۱
94  for number in numbers_list:
95      print(f"This loop: {multiple} * {number}: {multiple*number}")
96      multiple *= number # متغیر multiple برای عنصر در فهرست ضرب می‌شود
97
98  print("The multiple of the list is:", multiple)
99
```

Annotations in Persian are placed over the code to explain its behavior:

- Line 92: "فهرست نمونه‌ای از اعداد" (A sample list of numbers)
- Line 93: "متغیر برای ذخیره حاصل‌ضرب، مقدار اولیه ۱" (Variable for storing the product, initial value 1)
- Line 95: "متغیر multiple" (variable multiple)
- Line 96: "متغیر multiple برای عنصر در فهرست ضرب می‌شود" (multiple variable is used for each element in the list to calculate the product)

The terminal below shows the execution of the script:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/For_loops.py"
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
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

توضیحات کد:

۱. فهرستی از اعداد به نام `numbers` تعریف شده است.
۲. متغیر `multiple` با مقدار اولیه ۱ ایجاد شده است.
۳. در حلقه `for`، هر عنصر از `numbers` در `multiple` ضرب می‌شود.
۴. 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^{۵۰} 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 به‌ویژه در موقعی 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.

۳-۶) شیوه‌ی کار حلقه‌های تو در تو

فرض کنید از دو حلقه‌ی for استفاده کردایم، به‌طوری که 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 لیست مجزا استفاده کرد:

⁵⁰ Nested Loops

```

For_loops.py > ...
100
101 # ** Nested Loops **
102
103 for i in range(1, 4):
104     for j in range(1, 3):
105         print(f"i: {i}, j: {j}")
106
107
108
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\Python> python example & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/For_loops.py"
i: 1, j: 1
i: 1, j: 2
i: 2, j: 1
i: 2, j: 2
i: 3, j: 1
i: 3, j: 2
PS C:\Users\RezaSeyyed\Desktop\Python>

```

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 دوبعدی (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^{۵۱}" به عنوان واحدهای مستقلی از کد تعریف می‌شوند که وظایف مشخصی را بر عهده 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.

۱-۷) 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:

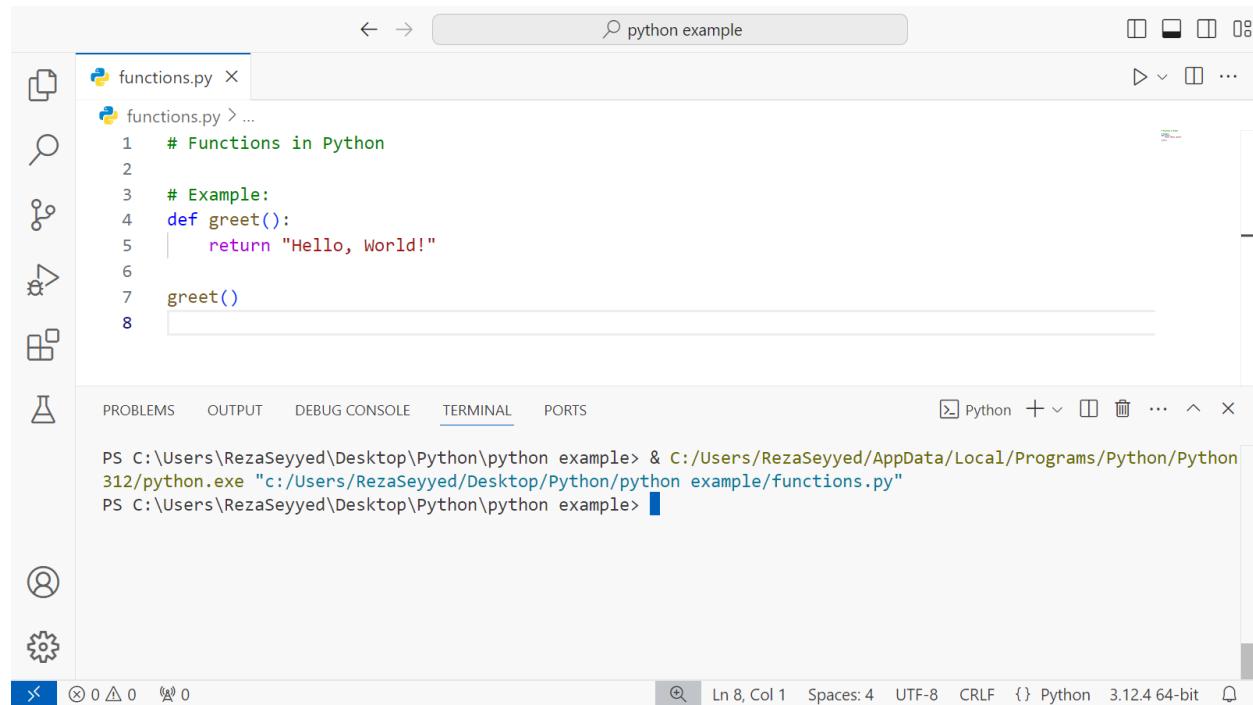
```
def double(number):
```

```
    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.

⁵¹ 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.

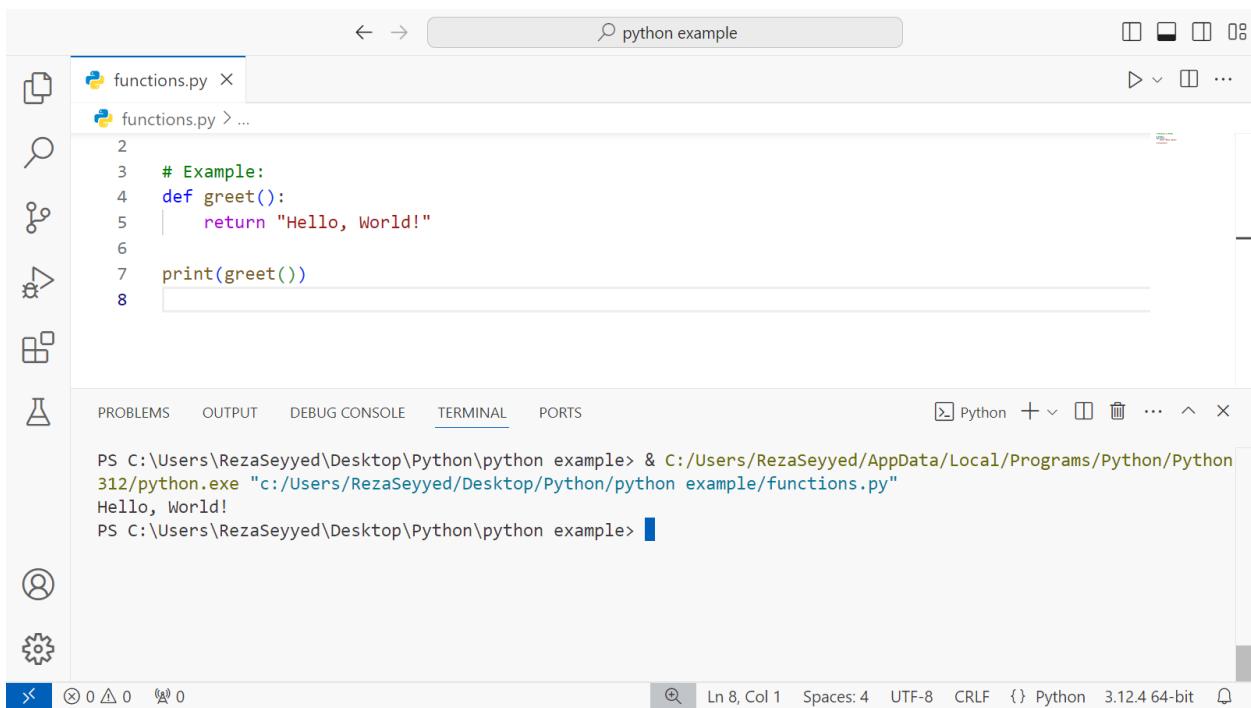


The screenshot shows the Visual Studio Code interface with the following details:

- File Explorer:** Shows a file named "functions.py".
- Code Editor:** Displays the following Python code:

```
1 # Functions in Python
2
3 # Example:
4 def greet():
5     return "Hello, World!"
6
7 greet()
```
- Terminal:** Shows the output of running the script:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```
- Status Bar:** Shows the current file is "functions.py", line 8, column 1, with 4 spaces, encoding is UTF-8, and the Python version is 3.12.4 64-bit.



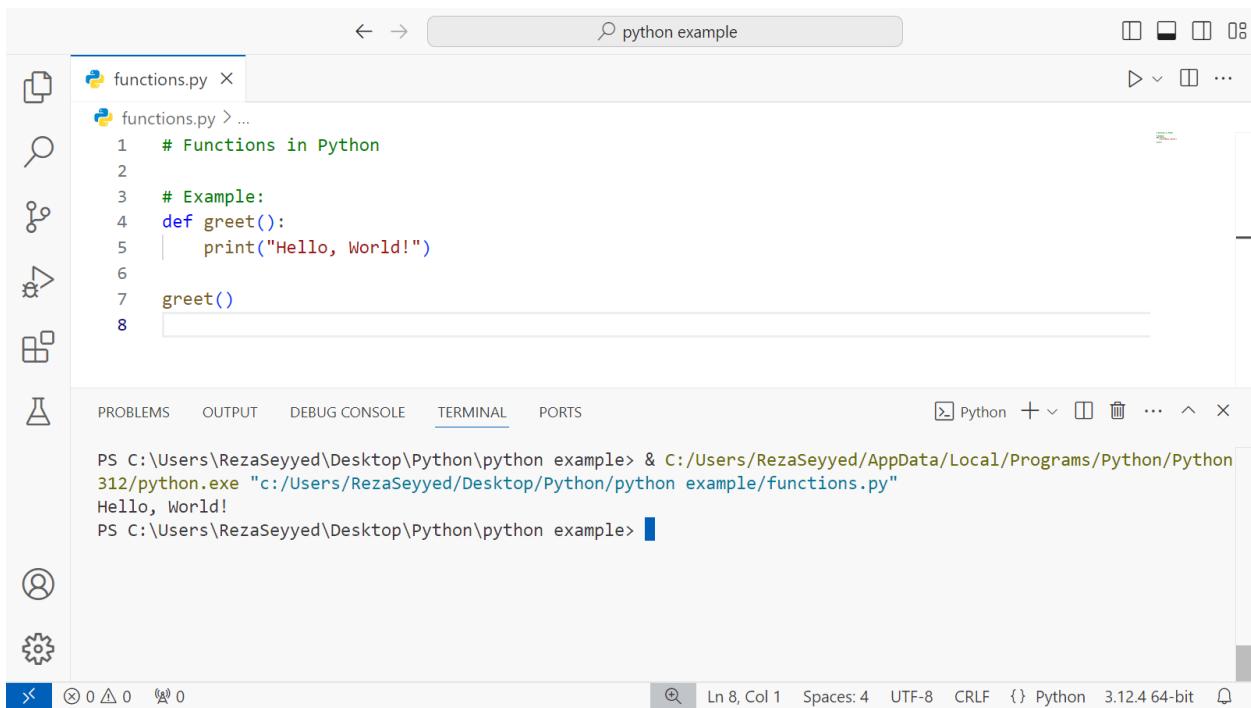
The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** python example
- Left Sidebar:** Shows icons for file operations (New, Open, Save, Find, Replace, Copy, Paste, Delete, Undo, Redo), a search icon, a refresh icon, a refresh icon, and a gear icon.
- Editor:** A code editor window titled "functions.py" containing the following Python code:

```
2
3 # Example:
4 def greet():
5     return "Hello, World!"
6
7 print(greet())
8
```
- Bottom Status Bar:** Shows file status (Ln 8, Col 1), spaces (Spaces: 4), encoding (UTF-8), line endings (CRLF), file type (Python), and version (3.12.4 64-bit).

In the terminal below the editor, the output of running the script is shown:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Hello, World!
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```



The screenshot shows the Visual Studio Code interface with the following details:

- Title Bar:** python example
- Left Sidebar:** Shows icons for file operations (New, Open, Save, Find, Replace, Copy, Paste, Delete, Undo, Redo), a search icon, a refresh icon, a refresh icon, and a gear icon.
- Editor:** A code editor window titled "functions.py" containing the following Python code:

```
1 # Functions in Python
2
3 # Example:
4 def greet():
5     print("Hello, World!")
6
7 greet()
8
```
- Bottom Status Bar:** Shows file status (Ln 8, Col 1), spaces (Spaces: 4), encoding (UTF-8), line endings (CRLF), file type (Python), and version (3.12.4 64-bit).

In the terminal below the editor, the output of running the script is shown:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Hello, World!
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

۲-۷) 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 خطا نیز آسان‌تر می‌شود.

۳-۷) توابع داخلی و توابع تعریف‌شده توسط کاربر

در پایتون، دو نوع کلی از توابع وجود دارد: "توابع داخلی" و "توابع تعریف‌شده توسط کاربر".

توابع داخلی پایتون توابعی هستند که به‌طور پیش‌فرض در زبان برنامه‌نویسی موجودند و برای انجام کارهای رایج مانند محاسبات ریاضی، کار با رشته‌ها و پردازش لیست‌ها استفاده می‌شوند؛ برای مثال `len()` برای محاسبه طول لیست یا رشته و `sum()` برای جمع‌کردن عناصر یک لیست. اما برنامه‌نویس می‌تواند تابع خود را نیز ایجاد کند تا وظایف خاصی را انجام دهد که در برنامه‌نویسی و حل مسائل پیچیده بسیار کاربرد دارد.

به‌طور خلاصه، "توابع" در پایتون ابزاری حیاتی و ضروری هستند که به برنامه‌نویسان امکان می‌دهند تا کدی تمیز، مؤثر و مقیاس‌پذیر بنویسند. توابع با جداسازی بخش‌های مختلف کد و افزایش امکان استفاده مجدد از آن، برنامه‌نویسی را تسهیل می‌کنند و به دستیابی به ساختارهای قابل مدیریت‌تر کمک می‌کنند.

۴-۷) نحوه اجرا و فراخوانی توابع

زمانی که تابعی با نام دلخواه ساختید، می‌توانید در خط جدید با فراخوانی نام تابع، از آن تابع استفاده کنید.

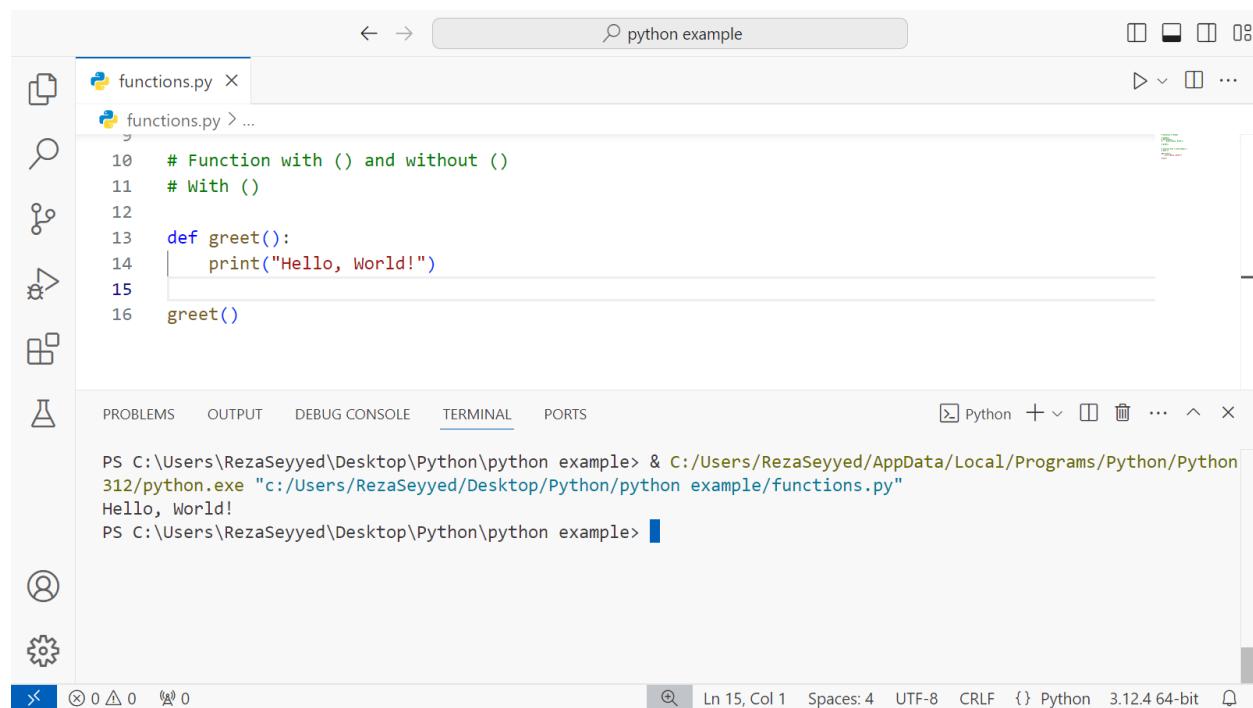
نکته: معمولاً زمانی که نام تابع همراه با پرانتز قرار می‌دهید، آن تابع را فرا می‌خوانید (در ادامه توضیحات بیشتری داده خواهد شد).

در زبان برنامه‌نویسی پایتون، "فراخوانی تابع" به دو شیوه متفاوت انجام می‌شود: "با پرانتز" و "بدون پرانتز". درک تفاوت میان این دو شیوه ضروری است، زیرا هر یک نتایج متفاوتی به همراه دارند و در موقعیت‌های خاص خود به کار می‌روند.

۱-۴-۷) فراخوانی تابع با پرانتز

هنگامی که تابعی را "با پرانتز" فراخوانی می‌کنید، 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.

برای مثال:



The screenshot shows the Visual Studio Code (VS Code) interface. On the left, there's a sidebar with icons for file operations like open, save, and search. The main area is a code editor with a tab labeled 'functions.py'. The code in the editor is:

```
10 # Function with () and without ()
11 # With ()
12
13 def greet():
14     print("Hello, World!")
15
16 greet()
```

Below the code editor is a terminal window titled 'Python' which shows the output of running the script:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Hello, World!
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

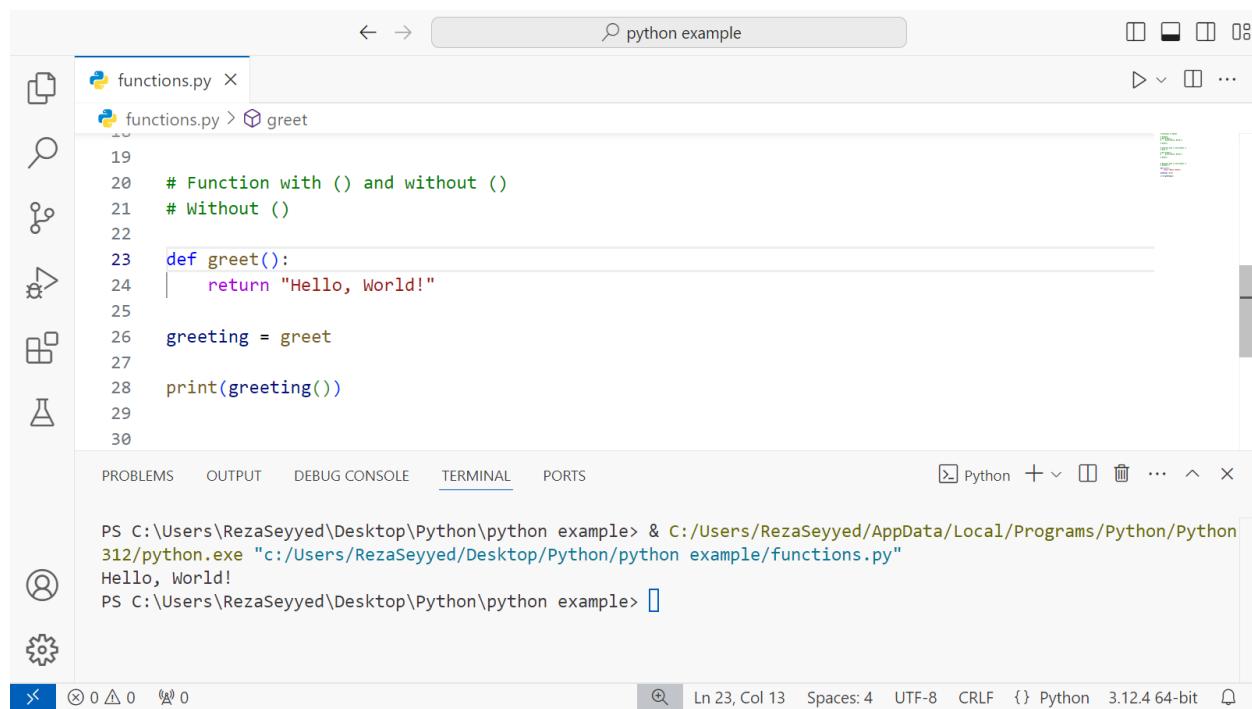
The bottom status bar indicates the file is 15 lines long, has 4 spaces, uses UTF-8 encoding, and is a Python 3.12.4 64-bit file.

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 می‌خواهیم تابع را بدون اجرای فوری، به متغیری اختصاص دهیم.

مثال زیر تفاوت میان این دو حالت را نشان می‌دهد:



```
functions.py > greet
19
20  # Function with () and without ()
21  # Without ()
22
23  def greet():
24      return "Hello, World!"
25
26  greeting = greet
27
28  print(greeting())
29
30
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Hello, World!
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

Ln 23, Col 13 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit

در این مثال، وقتی `greeting = greet` نوشته می‌شود، به تابع `greet` بدون اجرای آن اشاره می‌کنید. سپس با استفاده از `greet()` می‌توانید تابع را فراخوانی کرده و خروجی را دریافت کنید. اگر در همان ابتدا

به صورت خلاصه، اگر به متغیری، یک تابع با پرانتز داده شود، نتیجه آن در `greeting` نوشته می‌شد، تابع `greet` بلافصله اجرا می‌شود و نتیجه‌ی آن در `greeting = greet()` ذخیره می‌شد.

به صورت خلاصه، اگر به متغیری، یک تابع با پرانتز داده شود، نتیجه آن تابع در متغیر ذخیره می‌شود، اما اگر به 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

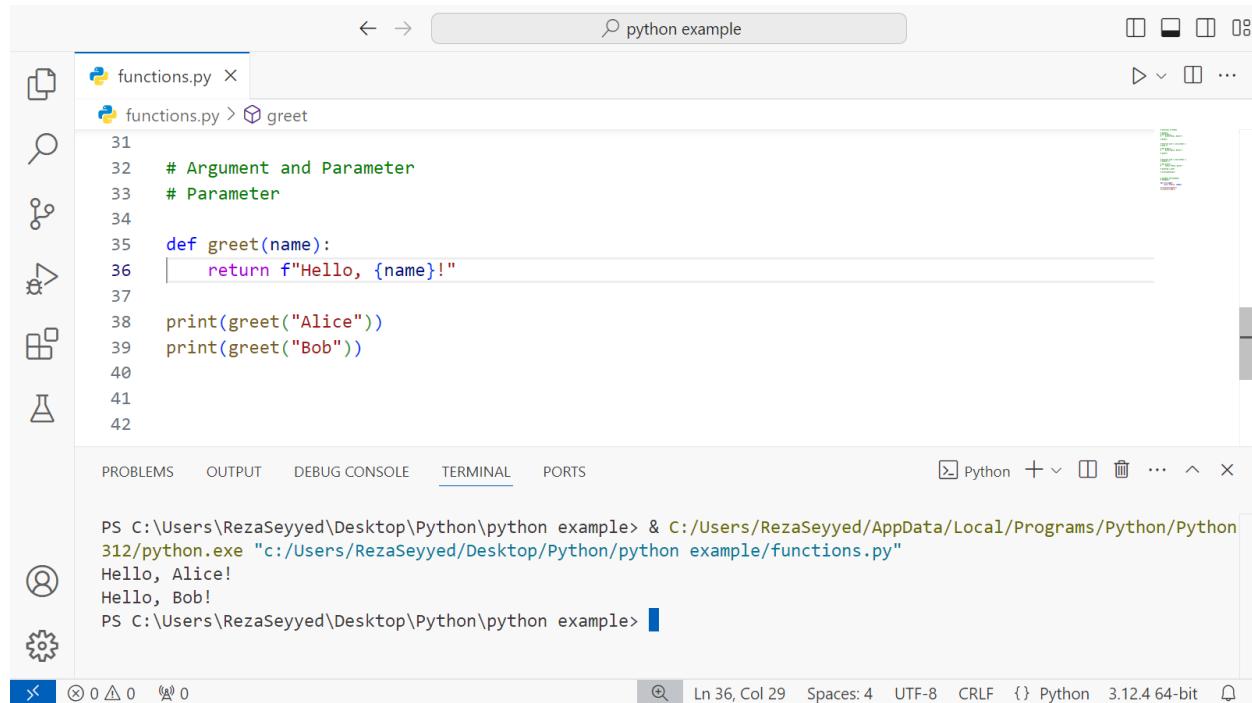
"پارامترها"^{۵۲} به عنوان ورودی‌های تابع عمل می‌کنند و به آن امکان می‌دهند تا با داده‌های مختلف کار کرده و رفتار خود را براساس نیاز تغییر دهد. در واقع، پارامترها متغیرهایی هستند که در هنگام تعریف تابع تعیین می‌شوند و مقادیری که تابع در زمان فراخوانی دریافت می‌کند، "آرگومان"^{۵۳} نامیده می‌شوند. این ویژگی 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.

⁵² Parameters

⁵³ 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، تابع زیر یک پارامتر به نام `name` می‌پذیرد و با توجه به مقدار دریافتی، پیامی شخصی‌سازی‌شده ایجاد می‌کند:



```
functions.py
31
32     # Argument and Parameter
33     # Parameter
34
35     def greet(name):
36         return f"Hello, {name}!"
37
38     print(greet("Alice"))
39     print(greet("Bob"))

PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Hello, Alice!
Hello, Bob!
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

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.

۷-۵-۲) انواع پارامترها در پایتون

پایتون امکان تعریف انواع مختلف پارامترها را فراهم می‌کند:

۱. پارامترهای موقعیتی^{۵۴}: این پارامترها بر اساس موقعیتی که به تابع ارسال می‌شوند مقداردهی می‌شوند. به عنوان مثال، در تابع زیر `x` و `y` به ترتیب با مقادیر `۱۰` و `۲۰` مقداردهی می‌شوند:

⁵⁴ Positional Parameters

```
functions.py
42 # Types of Parameters
43 # Positional Parameters
44
45 def add(x, y):
46     return x + y
47
48 print(add(10, 20))
49
50
51
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Python

```
ams/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
"
30
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

Ln 46, Col 20 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit

۲. پارامترهای کلیدوازه‌ای^{۵۵}: 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:

⁵⁵ Keyword Parameters

The screenshot shows a Microsoft Visual Studio Code (VS Code) interface. The top bar has a search field containing "python example". The left sidebar contains icons for file operations like Open, Save, Find, and Share. The main editor window displays a Python script named "functions.py". The code defines a function "introduce" that takes "name" and "age" parameters and returns a string. It then prints the result of calling this function with "age=25" and "name='Alice'". The bottom navigation bar shows tabs for PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL, and PORTS, with "TERMINAL" currently selected. The terminal window below shows the command "python functions.py" being run, followed by the output "My name is Alice and I am 25 years old." The status bar at the bottom shows file statistics (0 errors, 0 warnings, 0 info), line 57, column 39, and other system details.

```
50
51     # Types of Parameters
52     # Keyword Parameters
53
54     def introduce(name, age):
55         return f"My name is {name} and I am {age} years old."
56
57     print(introduce(age=25, name="Alice"))
58
59
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

ams/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"

My name is Alice and I am 25 years old.

PS C:\Users\RezaSeyyed\Desktop\Python\python example>

۳. پارامترهای پیشفرض^{۶۵}: این پارامترها دارای مقدار پیشفرضی هستند که اگر هنگام فراخوانی تابع مقداری به آنها اختصاص داده نشود، از مقدار پیشفرض استفاده می‌شود:

56 Default Parameters

The screenshot shows the Visual Studio Code (VS Code) interface. In the top left, there's a file tree icon. The main area displays a code editor with the file 'functions.py' open. The code contains a function definition:

```
60 # Types of Parameters
61 # Default Parameters
62
63 def greet(name="Guest"):
64     return f"Hello, {name}!"
65
66 print(greet())
67 print(greet("Alice"))
68
```

To the right of the code editor is a sidebar with various icons. Below the code editor are tabs for 'PROBLEMS', 'OUTPUT', 'DEBUG CONSOLE', and 'TERMINAL'. The 'TERMINAL' tab is active, showing the output of the Python interpreter:

```
Hello, Guest!
Hello, Alice!
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

The bottom status bar shows the file path 'C:\Users\RezaSeyyed\Desktop\Python\python example', the line number 'Ln 68, Col 1', and the character count 'Spaces: 4'. It also indicates the file is saved in 'UTF-8' encoding, uses 'CRLF' line endings, is a 'Python' file, and is a '3.12.4 64-bit' build.

۴. پارامترهای دلخواه^{۵۷}: زمانی که تعداد پارامترها نامشخص باشد، می‌توان از پارامترهای دلخواه استفاده کرد. با

افزودن '*' قبل از نام پارامتر، این نوع پارامتر به صورت یک لیست قابل دسترسی خواهد بود:

^{۵۷} Arbitrary Parameters

```
functions.py > ...
70  # Types of Parameters
71  # Arbitrary Parameters
72
73  def summarize(*args):
74      return sum(args)
75
76  print(f"Answer is: {summarize(1, 2, 3, 4)}")
77
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

Answer is: 10
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

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.

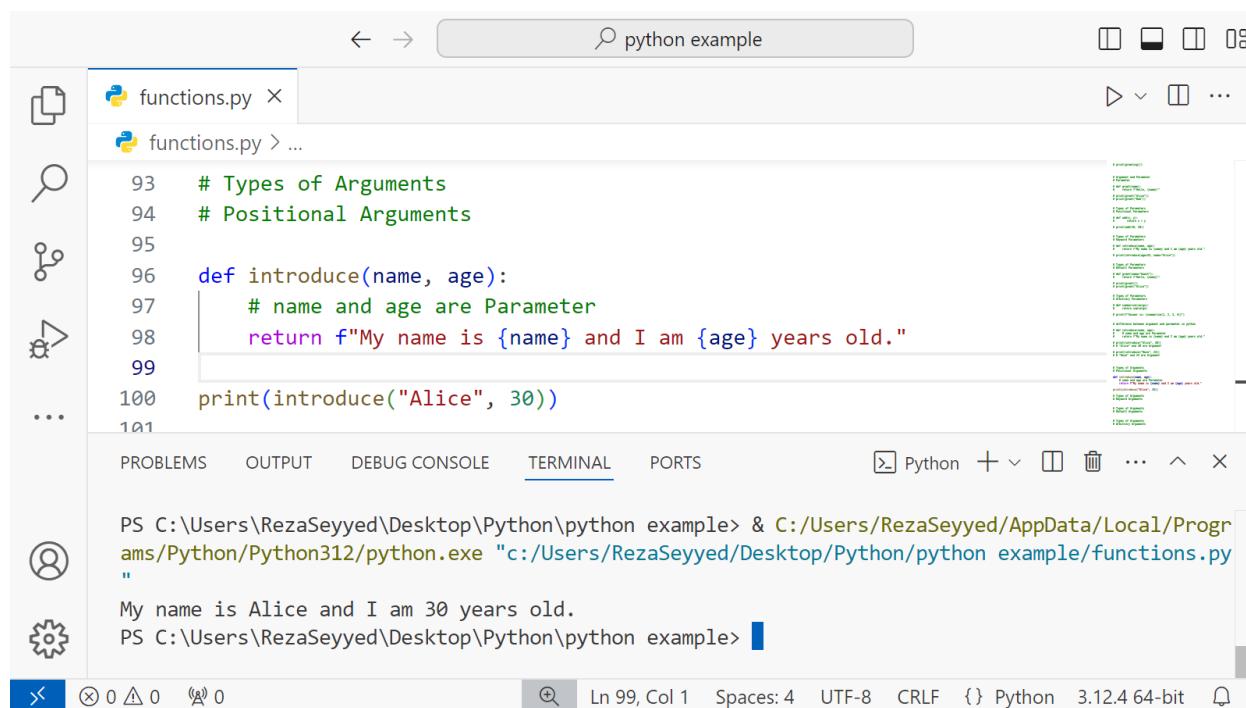
همان‌طور که اشاره شد، "پارامترها" متغیرهایی هستند که هنگام تعریف تابع مشخص می‌شوند و نقشی مانند "ورودی‌های تابع" را دارند. حال، "آرگومان‌ها" به مقادیری اطلاق می‌شود که در زمان "فراخوانی تابع" به این پارامترها اختصاص می‌یابد. به عبارت دیگر، پارامترها "ظرف‌هایی" هستند که در تعریف تابع تعیین می‌شوند، و آرگومان‌ها همان "محتویاتی" هستند که هنگام فراخوانی تابع، درون این ظرف‌ها قرار می‌گیرند.

۲-۵-۷) تعریف آرگومان

۱-۲-۵-۷) 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^{۵۸}: این آرگومان‌ها بر اساس موقعیت خود در زمان فراخوانی به پارامترهای تابع اختصاص داده می‌شوند. برای مثال:



```
# Types of Arguments
# Positional Arguments
def introduce(name, age):
    # name and age are Parameter
    return f"My name is {name} and I am {age} years old."
print(introduce("Alice", 30))
```

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
My name is Alice and I am 30 years old.
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

توضیحات: Alice به name و ۳۰ به age اختصاص می‌یابد.

^{۵۸} Positional Arguments

۲. آرگومان‌های کلیدواژه‌ای^{۵۹}: این آرگومان‌ها بر اساس نام پارامترها مقداردهی می‌شوند، که خوانایی کد را بهبود می‌بخشد و به ما امکان می‌دهند ترتیب آرگومان‌ها را تغییر دهیم:

```
functions.py
101
102 # Types of Arguments
103 # Keyword Arguments
104
105 def introduce(name, age):
106     # name and age are Parameter
107     return f"My name is {name} and I am {age} years old."
108
109 print(introduce(age=30, name="Alice"))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
"
My name is Alice and I am 30 years old.
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

۳. آرگومان‌های دلخواه^{۶۰}: گاهی تعداد آرگومان‌های ورودی معلوم نیست؛ در این صورت می‌توان با استفاده از این نوع آرگومان‌ها را به تابع ارسال کرد.

لیستی از آرگومان‌های موقعیتی و `**kwargs` دیکشنری‌ای از آرگومان‌های کلیدواژه‌ای را دریافت می‌کند:

⁵⁹ Keyword Arguments

⁶⁰ Arbitrary Arguments

```

functions.py > ...
111
112     # Types of Arguments
113     # Arbitrary Arguments
114
115     def print_info(*args, **kwargs):
116         print("Positional Arguments:", args)
117         print("Keyword Arguments:", kwargs)
118
119     print_info("Alice", 30, city="New York", profession="Engineer")
120
PROBLEMS    OUTPUT    DEBUG CONSOLE    TERMINAL    PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Positional Arguments: ('Alice', 30)
Keyword Arguments: {'city': 'New York', 'profession': 'Engineer'}
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

```

توضیحات: در این مثال، `Alice` و `30` آرگومان‌های موقعیتی و `city="New York"` و `profession="Engineer"` پارامترها به متغیرهایی اشاره دارند که هنگام تعریفتابع 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 (در ادامه بیشتر به این موضوع پرداخته می‌شود).

۳-۵-۷) مثال و تفاوت میان پارامترها و آرگومان‌ها

فرض کنید تابعی به نام `introduce` داریم که دو پارامتر `name` و `age` را می‌پذیرد. هنگام تعریف تابع، `name` و `age` به عنوان پارامترها شناخته می‌شوند. اما در زمان فراخوانی تابع، مقادیر خاصی به عنوان آرگومان به این پارامترها ارسال می‌شوند:

The screenshot shows the Visual Studio Code interface. In the top bar, there is a search field with the text "python example". Below it, the file "functions.py" is open, showing the following Python code:

```
79 # difference between argument and parameter in python
80
81 def introduce(name, age):
82     # name and age are Parameter
83     return f"My name is {name} and I am {age} years old."
84
85 print(introduce("Alice", 30))
86 # "Alice" and 30 are Argument
87
88 print(introduce("Reza", 24))
89 # "Reza" and 24 are Argument
```

In the terminal tab, the output is displayed:

```
My name is Alice and I am 30 years old.
My name is Reza and I am 24 years old.
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

در این مثال، "Alice" و "30" 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 کلیدواژه‌ای دلخواه.

*args) آرگومان‌های موقعیتی دلخواه با

هنگامی که در تعریفتابع 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 لیستی از ورودی‌ها دریافت می‌کند.

مثال زیر تابعی را نشان می‌دهد که چندین عدد را به عنوان ورودی دریافت می‌کند و جمع آن‌ها را محاسبه و بازمی‌گرداند:

The screenshot shows the Visual Studio Code interface with the following details:

- File Bar:** File, Edit, Selection, ...
- Search Bar:** python example
- Left Sidebar:** Includes icons for file operations (New, Open, Save, Find, Replace, Copy, Paste, Delete, Undo, Redo), a search icon, a branch icon, a refresh icon, and an ellipsis (...).
- Editor Area:** The file `functions.py` is open, containing the following code:

```
121
122     # Types of Arguments
123     # Arbitrary Arguments (*args explain)
124
125     def sum_numbers(*args):
126         total = sum(args)
127         return total
128
129     print(sum_numbers(1, 2, 3, 4))
130     print(sum_numbers(5, 10, 15))
131
```

A preview pane on the right shows the execution results of the code.
- Bottom Navigation:** PROBLEMS, OUTPUT, DEBUG CONSOLE, TERMINAL (underlined), PORTS
- Terminal:** Shows the command `PS C:\Users\RezaSeyyed\Desktop\Python\python example>` followed by the output `10` and `30`.
- Status Bar:** Shows 0 errors, 0 warnings, 0 info, and 0 hints. It also displays the current line (Ln 128, Col 1), spaces (Spaces: 4), encoding (UTF-8), line endings (CRLF), file type (Python), and version (3.12.4 64-bit).

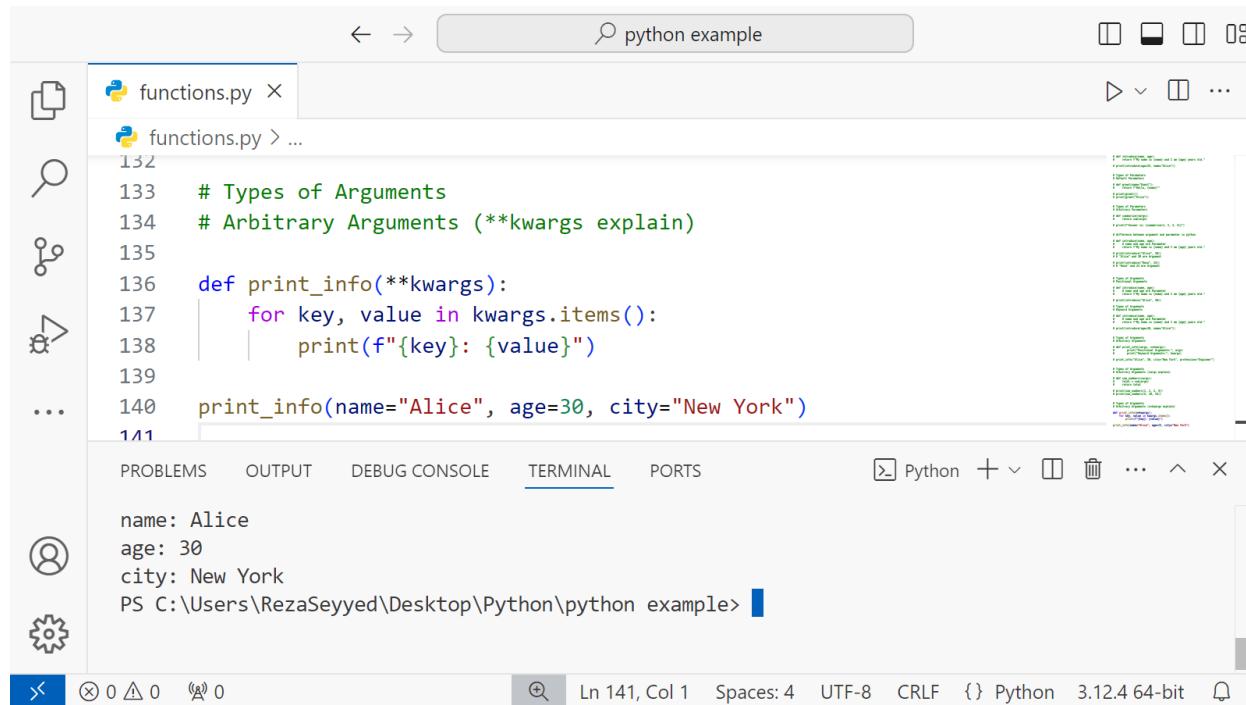
در این مثال، `args` یک تاپل شامل تمام آرگومان‌های ورودی است. تابع `sum_numbers` می‌تواند هر تعداد عدد را به عنوان ورودی، سذباد و مجموع آنها را محاسبه کند.

۷-۵-۴-۲) آ، گومان‌های، کلیدواژه‌ای، 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 کلیدوازه‌ای را دریافت کند.

به مثال زیر توجه کنید:



```
functions.py > ...
132
133 # Types of Arguments
134 # Arbitrary Arguments (**kwargs explain)
135
136 def print_info(**kwargs):
137     for key, value in kwargs.items():
138         print(f'{key}: {value}')
139
140 print_info(name="Alice", age=30, city="New York")
141
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

name: Alice
age: 30
city: New York
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

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 موقعيتی و هم آرگومان‌های کلیدوازه‌ای دلخواه را پذیرد. در این صورت، `*args` و `**kwargs` می‌توانند با هم ترکیب شوند؛ به شرط آنکه `*args` قبل از `**kwargs` در تعریف تابع قرار گیرد.

مثال زیر ترکیب این دو نوع آرگومان را نشان می‌دهد:

```
functions.py > display_info
143 # Types of Arguments
144 # Arbitrary Arguments (*args and **kwargs)
145
146 def display_info(*args, **kwargs):
147     print("Positional arguments:", args)
148     print("Keyword arguments:", kwargs)
149
150 display_info(1, 2, 3, name="Alice", age=30)
151
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Positional arguments: (1, 2, 3)
Keyword arguments: {'name': 'Alice', 'age': 30}
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

Ln 148, Col 40 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit

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

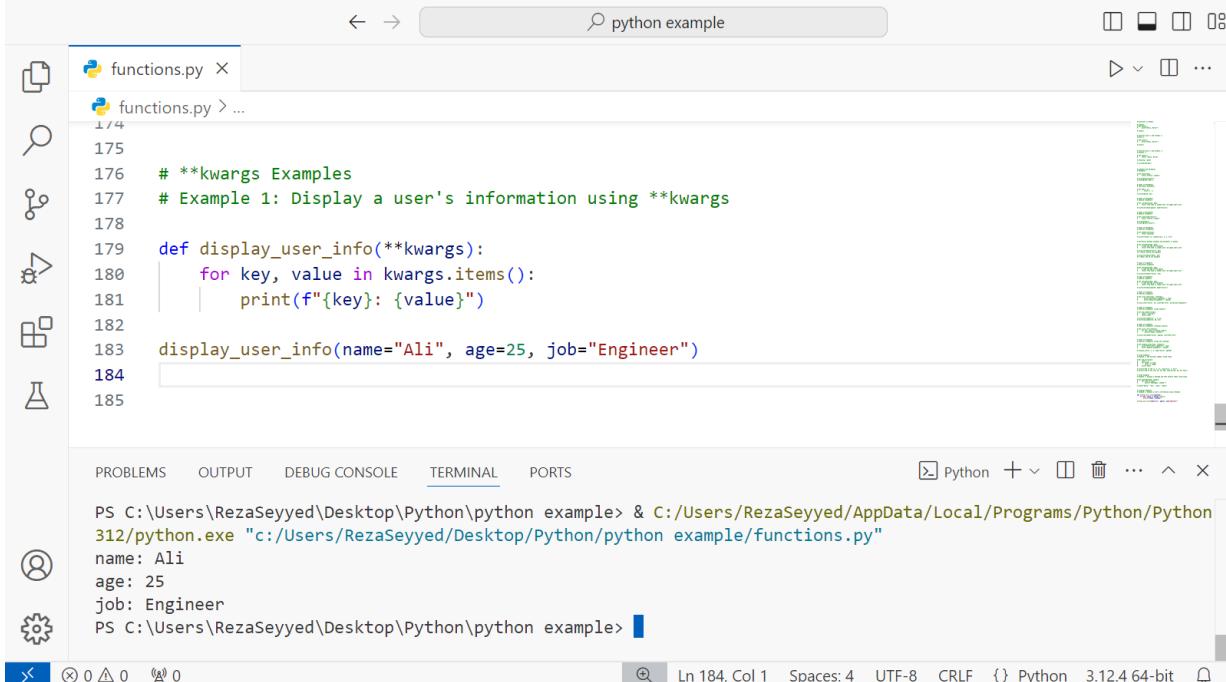
```
functions.py > ...
152
153     # *args Examples
154     # Example 1: Add multiple numbers using *args
155
156     def sum_all(*args):
157         total = 0
158         for number in args:
159             total += number
160
161         print(f"Sum of the (1, 2, 3): {sum_all(1, 2, 3)}")
162         print(f"Sum of the (10, 20, 30, 40): {sum_all(10, 20, 30, 40)}")
163
164
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Sum of the (1, 2, 3): 6
Sum of the (10, 20, 30, 40): 100
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

```

مثال ۲. نمایش یک پیغام و سپس چندین نام با استفاده از `*args`.

```
functions.py > ...
165
166     # *args Examples
167     # Example 2: Display a message and then several names using *args
168
169     def greet(message, *names):
170         for name in names:
171             print(f"{message}, {name}!")
172
173     greet("Hello", "Ali", "Sara", "Reza")
174
175
176
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Hello, Ali!
Hello, Sara!
Hello, Reza!
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

مثال ۳. نمایش اطلاعات یک کاربر با **kwargs



The screenshot shows a Python file named `functions.py` in the VS Code editor. The code defines a function `display_user_info` that takes `**kwargs` and prints each key-value pair. It then calls this function with parameters `name="Ali"`, `age=25`, and `job="Engineer"`. The terminal below shows the output: `name: Ali`, `age: 25`, and `job: Engineer`.

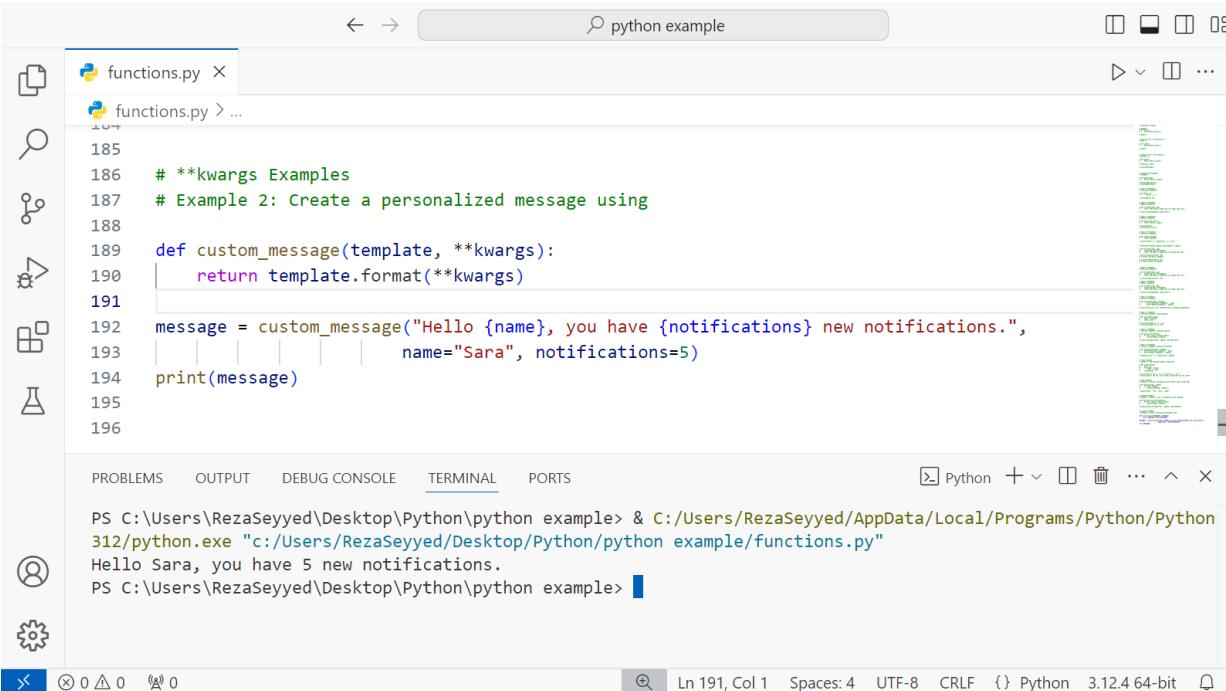
```
175
176 # **kwargs Examples
177 # Example 1: Display a user's information using **kwargs
178
179 def display_user_info(**kwargs):
180     for key, value in kwargs.items():
181         print(f"{key}: {value}")
182
183 display_user_info(name="Ali", age=25, job="Engineer")
184
185
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
name: Ali
age: 25
job: Engineer
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

Ln 184, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit

مثال ۴. ساخت یک پیام شخصی‌سازی شده با **kwargs



The screenshot shows a Python file named `functions.py` in the VS Code editor. The code defines a function `custom_message` that takes a template string and `**kwargs`. It then formats the template using `format(**kwargs)`. The code creates a message with `{name}` and `{notifications}` placeholders, sets `name` to "Sara" and `notifications` to 5, and prints the result. The terminal below shows the output: "Hello Sara, you have 5 new notifications."

```
185
186 # **kwargs Examples
187 # Example 2: Create a personalized message using
188
189 def custom_message(template, **kwargs):
190     return template.format(**kwargs)
191
192 message = custom_message("Hello {name}, you have {notifications} new notifications.",
193                           name="Sara", notifications=5)
194 print(message)
195
196
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Hello Sara, you have 5 new notifications.
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

Ln 191, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.4 64-bit

۶-۷ توابع درونی در زبان برنامه‌نویسی پایتون

"توابع درونی"^{۶۱} 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 توابع 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 توابع 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 `sum()` به شما اجازه می‌دهند نوع داده‌ها را به سادگی به اعداد صحیح، اعداد اعشاری و یا رشته تبدیل کنید. دیگر تابع مانند `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 تابع پیچیده و تکراری برای عملیات‌های رایج ندارد، که این موضوع بهینه‌سازی و خوانایی کد را افزایش می‌دهد.

این تابع، به سبب بهینه‌سازی و سرعت اجرایی بالا، برای برنامه‌هایی با حجم پردازش بالا نیز ایده‌آل‌اند و از طرفی با ترکیب آن‌ها با تابع دیگر یا در ساختارهای مختلف پایتون مانند لیست‌ها، دیکشنری‌ها، و تاپل‌ها، امکانات بیشتری برای پیاده‌سازی الگوریتم‌های پیچیده به برنامه‌نویسان ارائه می‌دهند. به طور خلاصه، توابع درونی پایتون

⁶¹ Built-in Functions

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 ارائه شده است.

این توابع جزء توابع اصلی پایتون هستند و در هر برنامه‌ای بدون نیاز به وارد کردن کتابخانه خاصی قابل استفاده‌اند.

توابع و آرگومان‌ها	توضیحات
abs(x)	مقدار مطلق عدد x را بازمی‌گرداند.
all(iterable)	بررسی می‌کند که آیا تمام مقادیر موجود در iterable درست هستند یا خیر (همه برابر True).
any(iterable)	بررسی می‌کند که آیا حداقل یکی از مقادیر موجود در iterable درست است یا خیر.
ascii(object)	نسخه‌ای قابل نمایش از شیء object بر می‌گرداند که فقط شامل کاراکترهای ASCII باشد.
bin(x)	رشته‌ای از نمایش دودویی (باینری) عدد صحیح x را بازمی‌گرداند.
bool([x])	مقدار x را به مقدار بولین (False یا True) تبدیل می‌کند.
bytearray([source, encoding, errors])	آرایه‌ای از بایت‌ها بر اساس source ورودی ایجاد می‌کند.
bytes([source, encoding, errors])	شیء بایتی از source ورودی ایجاد می‌کند.
callable(object)	بررسی می‌کند که آیا object قابل فراخوانی (مثل تابع) است یا خیر.
chr(i)	کاراکتر مربوط به مقدار عددی i در جدول Unicode را بازمی‌گرداند.
complex([real, imag])	یک عدد مختلط (complex number) ایجاد می‌کند.

<code>dict(**kwargs)</code>	یک دیکشنری جدید ایجاد می‌کند.
<code>divmod(a, b)</code>	تقسیم و باقی‌مانده تقسیم <code>a</code> و <code>b</code> را به صورت زوج (<code>q, r</code>) برمی‌گرداند.
<code>enumerate(iterable, start=0)</code>	شیء <code>enumerate</code> از <code>iterable</code> ایجاد می‌کند که ایندکس‌ها را نیز برمی‌گرداند.
<code>filter(function, iterable)</code>	مقادیری از <code>iterable</code> که تابع <code>function</code> برایشان <code>True</code> برمی‌گرداند، انتخاب می‌کند.
<code>float([x])</code>	عدد <code>x</code> را به یک عدد اعشاری تبدیل می‌کند.
<code>format(value, format_spec)</code>	قالب‌بندی <code>value</code> طبق <code>format_spec</code>
<code>frozenset([iterable])</code>	یک مجموعه (set) تغییرنپذیر از عناصر <code>iterable</code> می‌سازد.
<code>getattr(object, name, [default])</code>	مقدار ویژگی <code>name</code> در شیء <code>object</code> را بازمی‌گرداند.
<code>globals()</code>	دیکشنری‌ای از متغیرهای سطح گلوبال برنامه را بازمی‌گرداند.
<code>hasattr(object, name)</code>	بررسی می‌کند که آیا شیء <code>object</code> دارای ویژگی <code>name</code> است یا خیر.
<code>hash(object)</code>	مقدار هش (hash) شیء <code>object</code> را بازمی‌گرداند.
<code>hex(x)</code>	نمایش هگزا (هگزادسیمال) عدد <code>x</code> را به صورت رشته بازمی‌گرداند.
<code>id(object)</code>	شناسه یکتای شیء <code>object</code> در حافظه را بازمی‌گرداند.
<code>input([prompt])</code>	ورودی را از کاربر دریافت و آن را به صورت رشته بازمی‌گرداند.
<code>int([x, base])</code>	عدد <code>x</code> را به یک عدد صحیح (integer) تبدیل می‌کند.
<code>isinstance(object, classinfo)</code>	بررسی می‌کند که آیا شیء <code>object</code> نمونه‌ای از کلاس <code>classinfo</code> است یا خیر.
<code>len(s)</code>	طول دنباله یا تعداد عناصر موجود در <code>s</code> را بازمی‌گرداند.

<code>list([iterable])</code>	یک لیست جدید از iterable می‌سازد.
<code>locals()</code>	دیکشنری‌ای از متغیرهای محلی تابع فعلی را بازمی‌گرداند.
<code>map(function, iterable, ...)</code>	تابع function را بر روی همه عناصر iterable اعمال و نتایج را بازمی‌گرداند.
<code>max(iterable)</code>	بزرگ‌ترین مقدار موجود در iterable را بازمی‌گرداند.
<code>min(iterable)</code>	کوچک‌ترین مقدار موجود در iterable را بازمی‌گرداند.
<code>next(iterator, default)</code>	عنصر بعدی iterator را بازمی‌گرداند.
<code>oct(x)</code>	نمایش اکتال (بر مبنای ۸) عدد x را به صورت رشته بازمی‌گرداند.
<code>open(file, mode='r', ...)</code>	فایل مشخص شده در file را باز می‌کند.
<code>ord(c)</code>	کد Unicode کاراکتر c را بازمی‌گرداند.
<code>pow(base, exp[, mod])</code>	محاسبه $base^exp$ و باقی‌مانده‌گیری به کمک mod (در صورت وجود).
<code>print(objects, sep=' ', end='\n', ...)</code>	چاپ objects با جداکننده sep و پایان‌دهنده end.
<code>range(start, stop[, step])</code>	تولید یک دنباله عددی از start تا stop با فاصله‌های step.
<code>repr(object)</code>	نمایش رسمی شیء object که قابل اجراست، بازمی‌گرداند.
<code>reversed(seq)</code>	دنباله‌ای معکوس از seq را بازمی‌گرداند.
<code>round(number[, ndigits])</code>	گرد کردن عدد number به تعداد اعشار ndigits.
<code>set([iterable])</code>	مجموعه‌ای از عناصر iterable می‌سازد.
<code>sorted(iterable)</code>	دنباله‌ای مرتب شده از iterable را بازمی‌گرداند.
<code>str(object)</code>	رشته‌ای از object می‌سازد.

<code>sum(iterable[, start=0])</code>	جمع عناصر موجود در iterable با مقدار اولیه start.
<code>tuple([iterable])</code>	تاپل جدیدی از iterable می‌سازد.
<code>type(object)</code> <code>type(name, bases, dict)</code>	نوع شیء object را بازمی‌گرداند یا یک کلاس جدید ایجاد می‌کند.
<code>zip(*iterables)</code>	شیء zip که عناصر iterables را به صورت زوج‌های مرتب ترکیب می‌کند، بازمی‌گرداند.

این جدول نمای کلی از توابع درونی پایتون به همراه آرگومان‌های معمولشان را نشان می‌دهد. برای جزئیات بیشتر درباره هر کدام، می‌توانید از تابع `help()` یا مستندات رسمی پایتون 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 به‌گونه‌ای طراحی شده‌اند که به‌سادگی و با سرعت بتوان تابع 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'` به‌دنبال 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 توابع 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.

```
functions.py > ...
197
198 # ** Lambda Function **
199
200 # Example 1: توان دوم اعداد فهرست
201 numbers = [1, 2, 3, 4, 5]
202 squared = list(map(lambda x: x**2, numbers))
203
204 print(squared)
205

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS Python + ... ^ x
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
[1, 4, 9, 16, 25]
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

این قطعه کد از تابع درونی `map` و عبارت `lambda` استفاده می‌کند تا توان دوم تمام اعداد موجود در لیست `numbers` را محاسبه کند. سپس نتیجه به لیستی جدید تبدیل شده و چاپ می‌شود.

تحلیل کد:

۱. تعریف لیست اعداد اولیه:

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

این خط لیستی 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 لیست `numbers` اعمال می‌کند.

`list()`: خروجی تابع `map` یک شیء قابل پیمایش (`map object`) است. برای تبدیل آن به

لیست، از تابع `list()` استفاده شده است.

۳. چاپ نتیجه:

```
print(squared)
```

این خط، لیست جدید حاوی توان دوم هر عدد از لیست اولیه را چاپ می‌کند.

توضیحات:

- عدد `۱` به توان دوم: $(1^2 = 1)$

- عدد `۲` به توان دوم: $(2^2 = 4)$

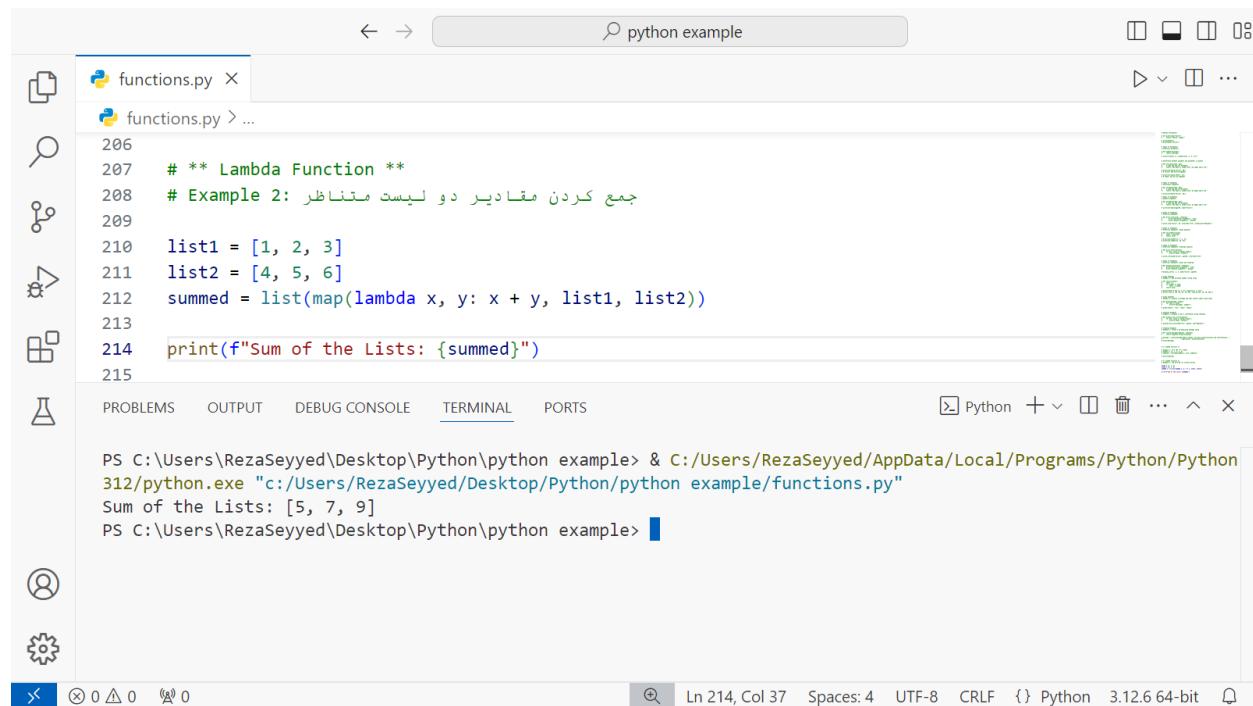
- عدد `۳` به توان دوم: $(3^2 = 9)$

- عدد `۴` به توان دوم: $(4^2 = 16)$

- عدد `۵` به توان دوم: $(5^2 = 25)$

این قطعه کد نمونه‌ای ساده و کاربردی از ترکیب `map` و `lambda` در پایتون است که بدون نیاز به تعریف یک تابع جداگانه، عملیات محاسباتی را انجام می‌دهد.

مثال ۲. با استفاده از `lambda`، مقادیر دو فهرست را با هم دیگر جمع کنید.



```
functions.py X
functions.py > ...

206
207 # ** Lambda Function **
208 # Example 2: جمع کردن مقادیر دو لیست متناظر
209
210 list1 = [1, 2, 3]
211 list2 = [4, 5, 6]
212 summed = list(map(lambda x, y: x + y, list1, list2))
213
214 print(f"Sum of the Lists: {summed}")
215

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + □ ... ^ ×

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Sum of the Lists: [5, 7, 9]
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

این قطعه کد از تابع `map` و عبارت `lambda` برای جمع کردن عناصر متناظر دو لیست `list1` و `list2` استفاده می‌کند. نتیجه‌ی این عملیات به صورت یک لیست جدید ذخیره و چاپ می‌شود.

تحلیل کد:

۱. تعریف دو لیست اولیه:

```
list1 = [1,2,3]
```

```
list2 = [4,5,6]
```

این دو لیست 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 لیست 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 لیست تبدیل می‌شود.

۳. چاپ نتیجه:

```
print(summed)
```

این خط، لیست حاوی مجموع عناصر متناظر دو لیست را چاپ می‌کند.

توضیحات جمع:

$$1 + 4 = 5 \rightarrow$$

$$2 + 5 = 7 \rightarrow$$

$$3 + 6 = 9 \rightarrow$$

نکات:

- ✓ این روش بسیار مفید است برای زمانی که نیاز دارید دو لیست یا بیشتر را به صورت موازی پردازش کنید، بدون نیاز به نوشتتن حلقه‌های اضافی. تابع `map` همراه با `lambda` کدی خواناتر و فشرده‌تر تولید می‌کند.
- ✓ با استفاده از `lambda`، فهرست میوه‌ها را براساس تعداد حروفشان مرتب‌سازی کنید.

```

functions.py
217 # ** Lambda Function **
218 # Example 3: مرتب‌سازی لیستی از رشته‌ها بر اساس طول آن‌ها
219
220 words = ['apple', 'banana', 'kiwi', 'grape']
221 sorted_words = sorted(words, key=lambda x: len(x))
222
223 print(f"Sorted List: {sorted_words}")
224
225
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Sorted List: ['kiwi', 'apple', 'grape', 'banana']
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

```

این کد از تابع درونی `sorted` و عبارت `lambda` برای مرتب‌سازی لیست `words` بر اساس طول رشته‌ها استفاده می‌کند. نتیجه‌ی این مرتب‌سازی به ترتیب صعودی طول رشته‌ها ذخیره و چاپ می‌شود.

تحلیل کد:

۱. تعریف لیست اولیه:

```
words = ['apple', 'banana', 'kiwi', 'grape']
```

لیستی 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 لیستی مرتب شده بر اساس یک معیار خاص بازمی‌گرداند، بدون اینکه لیست

اصلی تغییر کند.

▪ `key` : این پارامتر مشخص می‌کند که چگونه عناصر لیست باید مقایسه شوند.

▪ `lambda x: len(x)` : یک عبارت `lambda` است که برای هر عنصر `x` از لیست، 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، لیست 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 لیست 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.

۸-۷ آشنایی با مهم‌ترین توابع درونی `map()`, `filter()`, `zip()` در پایتون

۸-۸-۱) تابع درونی `zip()`

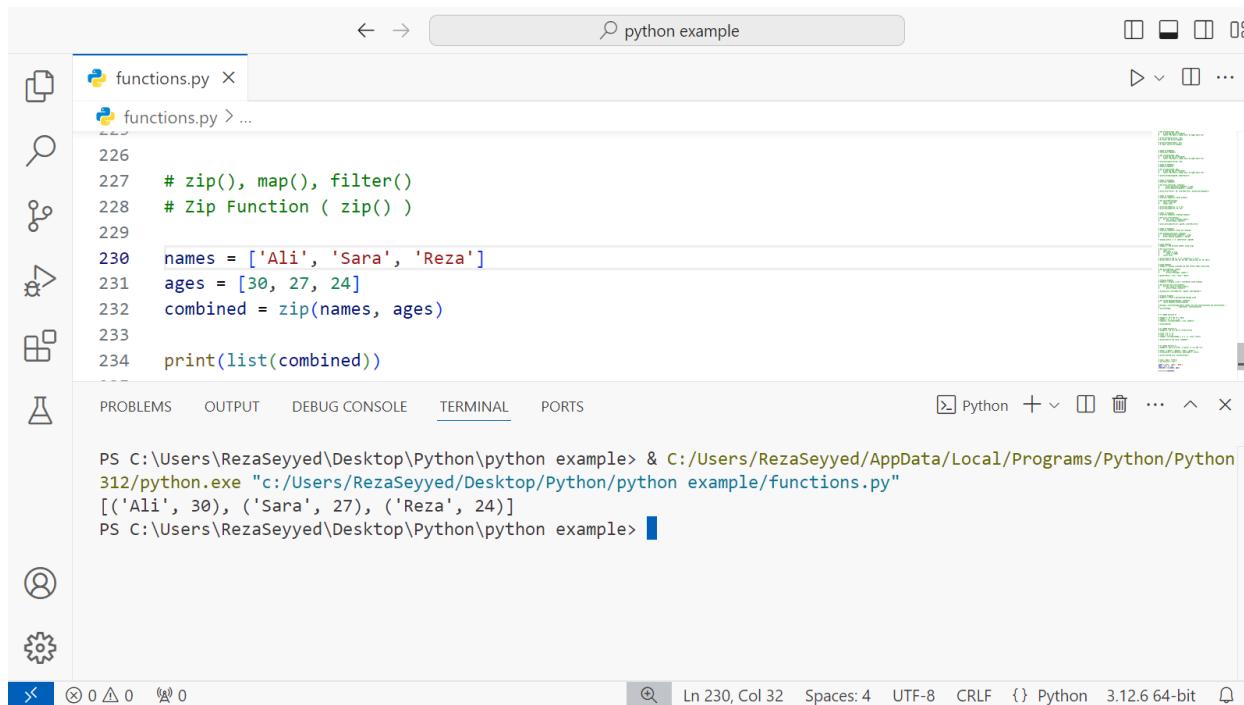
تابع درونی `zip` در پایتون یکی از توابع کاربردی برای ترکیب عناصر چندین "دنباله" (مانند لیست‌ها، تاپل‌ها و غیره) به صورت "زوج‌های مرتب" است. این تابع با قرار دادن عناصر مربوطه از هر دنباله در کنار هم، "شیء جدیدی ایجاد می‌کند که می‌تواند به صورت یک دنباله از "تاپل‌ها" پیمایش شود.

ساختار تابع `zip` به صورت زیر است:

`zip(*iterables)`

در این ساختار، `zip` می‌تواند شامل هر تعداد دنباله (مانند لیست‌ها 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:



```
functions.py > ...
226
227     # zip(), map(), filter()
228     # Zip Function ( zip() )
229
230     names = ['Ali', 'Sara', 'Reza']
231     ages = [30, 27, 24]
232     combined = zip(names, ages)
233
234     print(list(combined))

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
```

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
[('Ali', 30), ('Sara', 27), ('Reza', 24)]
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

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 لیست یا دیکشنری تبدیل شود، می‌توان از توابع `list()` یا `dict()` به صورت `list(zip(...))` و `dict(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 لیست، 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 آن که نیاز به نوشتتن حلقه باشد.

ساختار تابع `map` به صورت زیر است:

map(function, iterable)

در این ساختار، `function` تابعی است که می‌خواهد روی عناصر `iterable` اعمال کنید. این تابع باید به عنوان ورودی، یک عنصر از دنباله را دریافت کرده و یک نتیجه را برگرداند.

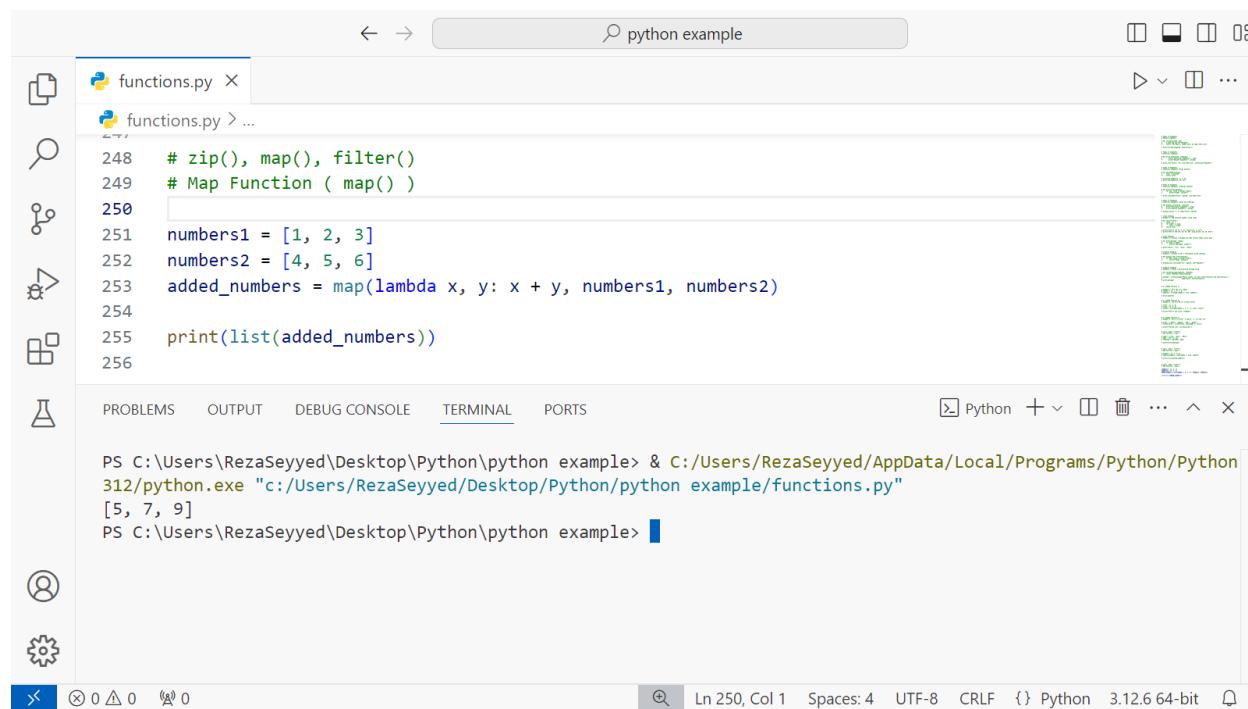
به عنوان مثال، فرض کنید می‌خواهید لیستی از اعداد را به توان دو برسانید. می‌توانید از `map` استفاده کنید تا تابع `lambda` این کار را روی همه عناصر اعمال کند:

```
functions.py X
functions.py > ...
237
238  # zip(), map(), filter()
239  # Map Function ( map() )
240
241  numbers = [1, 2, 3, 4]
242  squared_numbers = map(lambda x: x**2, numbers)
243
244  print(list(squared_numbers))
245

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python + ⌂ ... ^ ×
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
[1, 4, 9, 16]
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

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 مشاهده نتایج، معمولاً آن را به یک نوع داده‌ی مشخص، مانند لیست، 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.



The screenshot shows the Visual Studio Code (VS Code) interface. On the left is the file tree, showing a single file named "functions.py". The main area contains the following Python code:

```
248 # zip(), map(), filter()
249 # Map Function ( map() )
250
251 numbers1 = [1, 2, 3]
252 numbers2 = [4, 5, 6]
253 added_numbers = map(lambda x, y: x + y, numbers1, numbers2)
254
255 print(list(added_numbers))
256
```

Below the code editor is the terminal window, which shows the output of running the script:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
[5, 7, 9]
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

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 filter() تابع درونی

تابع درونی `filter` در پایتون ابزاری کاربردی برای "فیلتر کردن عناصر" یک "دبالة" (مانند لیست، تاپل یا مجموعه) بر اساس یک شرط مشخص است. این تابع برای موقعی ایدهآل است که می‌خواهد تنها "عناصری" از یک دبالة 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 `function` که `iterable` برای آن‌ها مقدار True بگرداند، در نتیجه‌ی نهایی قرار می‌گیرند؛ بقیه عناصر حذف خواهند شد. در صورت استفاده از `None` به جای برگرداند، در نتیجه‌ی نهایی قرار می‌گیرند؛ بقیه عناصر حذف خواهند شد. در صورت استفاده از `None` به جای True، تابع `filter` تنها عناصری که خودشان به طور طبیعی برابر با True هستند (یعنی مقدار صفر، None و False نیستند) را نگه می‌دارد.

به عنوان مثال، اگر بخواهیم از یک فهرست، اعداد زوج و فرد را جدا کنیم، می‌توانیم از تابع `filter` با استفاده از تابع `lambda` به شکل زیر استفاده کنیم:

```
258 # zip(), map(), filter()
259 # Filter Function ( filter() )
260
261 numbers = [1, 2, 3, 4, 5, 6]
262 even_numbers = filter(lambda x: x % 2 == 0, numbers)
263 odd_numbers = filter(lambda x: x%2 != 0, numbers)
264
265 print(f"Even numbers: {list(even_numbers)}")
266 print(f"Odd numbers: {list(odd_numbers)}")
267
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/functions.py"
Even numbers: [2, 4, 6]
Odd numbers: [1, 3, 5]
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

در `x % 2 == 0`، تابع `filter`، تنها آن عناصری از `numbers` را انتخاب کرد که شرط `x % 2 == 0` برایشان برقرار است، یعنی اعداد زوج.

در `x % 2 != 0`، 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 لیست، 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) خطاها و انواع آن‌ها در زبان برنامه‌نویسی پایتون

در فرآیند توسعه نرم‌افزار، خطاها^{۶۲} بخش جدایی‌ناپذیری از برنامه‌نویسی هستند. پایتون، به عنوان یک زبان سطح بالا، ابزارهای متعددی برای شناسایی و مدیریت خطاها ارائه می‌دهد. این خطاها عموماً به سه دسته اصلی تقسیم می‌شوند: خطاهاي ساختاري^{۶۳}، خطاهاي زمان اجرا^{۶۴}، و خطاهاي منطقى^{۶۵}. در ادامه، هر یک از این خطاها را بررسی می‌کنیم.

۱-۸) خطاهاي نحوی

این دسته از خطاها 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 خطاهاي نحوی، برنامه پیش از اجرا متوقف می‌شود.

مثال:

⁶² Errors

⁶³ Syntax Errors

⁶⁴ Runtime Errors

⁶⁵ Logical Errors

The screenshot shows the Visual Studio Code (VS Code) interface. On the left is the sidebar with icons for files, search, and other tools. The main area shows a Python file named 'Errors_Handling.py'. The code contains several syntax errors, such as missing colons and commas. A tooltip from the 'Python' extension is open, providing details about the current process ID (1932), command line, and shell integration status. Below the editor is the 'TERMINAL' tab, which displays the output of running the script. It shows the command being run, the file path, and the specific syntax error at line 13, which is 'SyntaxError: '(' was never closed'. The bottom status bar shows the file path, line number (Ln 13, Col 20), and other settings.

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 نحوی، برنامه تا رسیدن به خطای موردنظر

اجرا می‌شود، اما در آن لحظه متوقف می‌گردد.

مثال رایج:

تقسیم بر صفر:

A screenshot of the Visual Studio Code interface. The left sidebar shows icons for file operations, search, navigation, and other tools. The main editor window displays a Python script named 'Errors_Handling.py' with the following code:

```
15
16 # Errors and Errors Handling
17
18 # Types of Errors
19     # Syntax Errors
20     # Runtime Errors
21     # Logical Errors
22
23 # Runtime Errors
24
25 print(100/0)
```

The terminal below the editor shows the output of running the script:

```
312\python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"
Traceback (most recent call last):
  File "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py", line 25, in <module>
    print(100/0)
           ~~~^~
ZeroDivisionError: division by zero
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

دسترسی به شاخص نامعتبر در لیست:

A screenshot of the Visual Studio Code interface. The left sidebar shows icons for file operations, search, navigation, and other tools. The main editor window displays a Python script named 'Errors_Handling.py' with the following code:

```
17
18 # Types of Errors
19     # Syntax Errors
20     # Runtime Errors
21     # Logical Errors
22
23 # Runtime Errors
24
25 # print(100/0)
26
27 my_list = [1, 2, 3]
28 print(my_list[5])
29
```

The terminal below the editor shows the output of running the script:

```
312\python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"
Traceback (most recent call last):
  File "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py", line 28, in <module>
    print(my_list[5])
           ~~~~~^~~
IndexError: list index out of range
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

حل خطأ:

این خطاهای با استفاده از مدیریت استثنای (try/except) و بررسی دقیق شرایط قبل از اجرای کد قابل حل هستند.

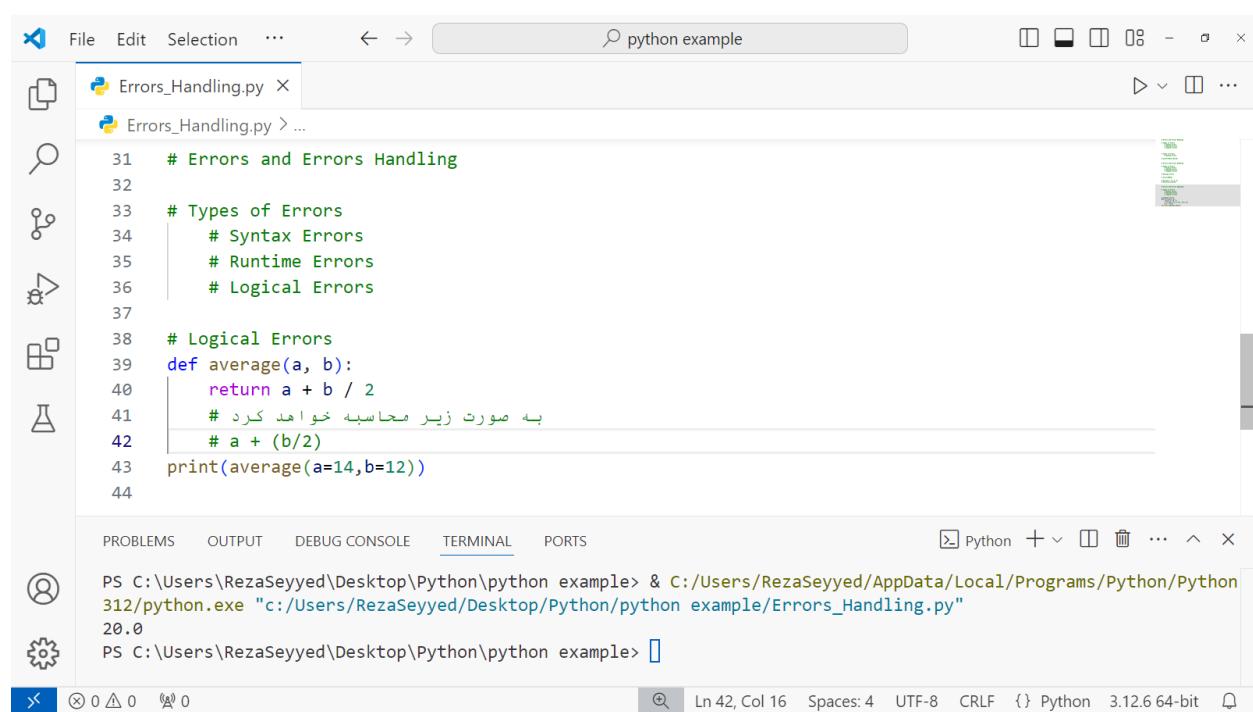
۳-۸ خطاهای منطقی

این خطاهای زمانی رخ می‌دهند که برنامه بدون توقف اجرا می‌شود، اما نتیجه‌های که تولید می‌کند، نادرست است.

خطاهای منطقی معمولاً ناشی از اشتباهات برنامه‌نویس در طراحی منطق کد هستند.

مثال:

فرض کنید قصد محاسبه میانگین دو عدد را داری:



The screenshot shows a code editor window for a file named 'Errors_Handling.py'. The code contains a function 'average' that calculates the average of two numbers. There is a logical error in the code where it prints the result of the addition of 'a' and 'b' instead of their average. The terminal below shows the execution of the script and the resulting output of 20.0, which is incorrect.

```
# Errors and Errors Handling
# Types of Errors
# Syntax Errors
# Runtime Errors
# Logical Errors
# Logical Errors
def average(a, b):
    return a + b / 2
# صورت زیر محاسبه خواهد کرد
# a + (b/2)
print(average(a=14,b=12))
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"

20.0

PS C:\Users\RezaSeyyed\Desktop\Python\python example>

در اینجا اولویت عملگرها نادیده گرفته شده و میانگین نادرست محاسبه می‌شود.

حل خطأ:

The screenshot shows a Python code editor interface. The file 'Errors_Handling.py' contains the following code:

```
37
38 # Logical Errors
39 def average(a, b):
40     return a + b / 2
41     # صورت زیر محاسبه خواهد کرد
42     # a + (b/2)
43 print(f"First Ans: {average(a=14,b=12)}")
44
45 # Solve Problem
46 def average(a, b):
47     return (a + b) / 2
48     # صورت زیر محاسبه خواهد کرد
49     # (a + b) / 2
50 print(f"Second Ans: {average(a=14,b=12)}")
```

The terminal output shows:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"
First Ans: 20.0
Second Ans: 13.0
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

۴-۸) سایر خطاهای متداول در پایتون

پایتون به دلیل غنی بودن در مدیریت خطا، مجموعه‌ای از خطاهای از پیش تعریف شده را ارائه می‌دهد که برخی

از آن‌ها عبارت‌اند از:

. زمانی که عملیاتی بر روی نوع داده نامناسبی انجام شود. **TypeError** ♦

A screenshot of the Visual Studio Code interface. The title bar says "python example". The left sidebar shows files "Errors_Handling.py" and "Errors_Handling.py > ...". The main editor area contains the following Python code:

```
53     # Other Errors
54     # TypeErrors
55
56     def Sum_Operation(a, b):
57         return f"Sum of a , b: {a + b}"
58
59     print(Sum_Operation("12", 4))
60
61
62
63
```

The terminal tab is active, showing the output of running the script:

```
File "c:\Users\RezaSeyyed\Desktop\Python\python example\Errors_Handling.py", line 59, in <module>
    print(Sum_Operation("12", 4))
               ^^^^^^^^^^^^^^^^^^
File "c:\Users\RezaSeyyed\Desktop\Python\python example\Errors_Handling.py", line 57, in Sum_Operation
    return f"Sum of a , b: {a + b}"
               ~~~^~
TypeError: can only concatenate str (not "int") to str
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

The status bar at the bottom shows "Ln 55, Col 1" and "Python 3.12.6 64-bit".

زمانی که مقدار داده ورودی برای یک عملیات مناسب نباشد. **ValueError** ❖

A screenshot of the Visual Studio Code interface. The title bar says "python example". The left sidebar shows files "Errors_Handling.py" and "Errors_Handling.py > ...". The main editor area contains the following Python code:

```
60
61
62     # Other Errors
63     # ValueErrors
64
65     def Value_Error_Example(a):
66         return f"{int(a)}"
67
68     print(Value_Error_Example("Hello World!"))
```

The terminal tab is active, showing the output of running the script:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"
Traceback (most recent call last):
  File "c:/Users/RezaSeyyed/Desktop/Python/python example\Errors_Handling.py", line 68, in <module>
    print(Value_Error_Example("Hello World!"))
               ^^^^^^^^^^^^^^^^^^
  File "c:/Users/RezaSeyyed/Desktop/Python/python example\Errors_Handling.py", line 66, in Value_Error_Example
    return f"{int(a)}"
               ^^^^
ValueError: invalid literal for int() with base 10: 'Hello World!'
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

The status bar at the bottom shows "Ln 68, Col 43" and "Python 3.12.6 64-bit".

زمانی که به یک متغیر یا نام تعریف نشده دسترسی پیدا کنید. **NameError** ♦♦

The screenshot shows a Python code editor interface with the following details:

- File Bar:** File, Edit, Selection, ..., python example
- Left Sidebar:** Shows icons for file operations like Open, Save, Find, and others.
- Code Editor:** Displays a file named "Errors_Handling.py" with the following content:

```
71 # Other Errors
72 # NameErrors
73
74 # میج متغیری یا تابعی تعریف نکردیم
75 print(f"value of x is {x}")
```
- Terminal:** Shows the command PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py". The output shows a Traceback error: NameError: name 'x' is not defined.
- Status Bar:** PROBLEMS 1, OUTPUT, DEBUG CONSOLE, TERMINAL, PORTS, Python, Ln 75, Col 28, Spaces: 4, UTF-8, CRLF, Python 3.12.6 64-bit.

زمانی که کلیدی وجود ندارد اما از دیکشنری درخواست شود. **KeyError** ♦♦

ImportError: زمانی که یک مازول یا نام در مازول یافت نشود.

The screenshot shows a Python file named `Errors_Handling.py` open in VS Code. The code attempts to import a non-existent module:

```
91
92     # Other Errors
93     |     # ImportErrors
94
95     import nonexistent_module
96
97     print(help(nonexistent_module))
```

The line `import nonexistent_module` is underlined with a red squiggle, indicating a syntax error. The terminal below shows the execution of the script and the resulting traceback:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"
Traceback (most recent call last):
  File "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py", line 95, in <module>
    import nonexistent_module
ModuleNotFoundError: No module named 'nonexistent_module'
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

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 (Debugging).

فصل نهم

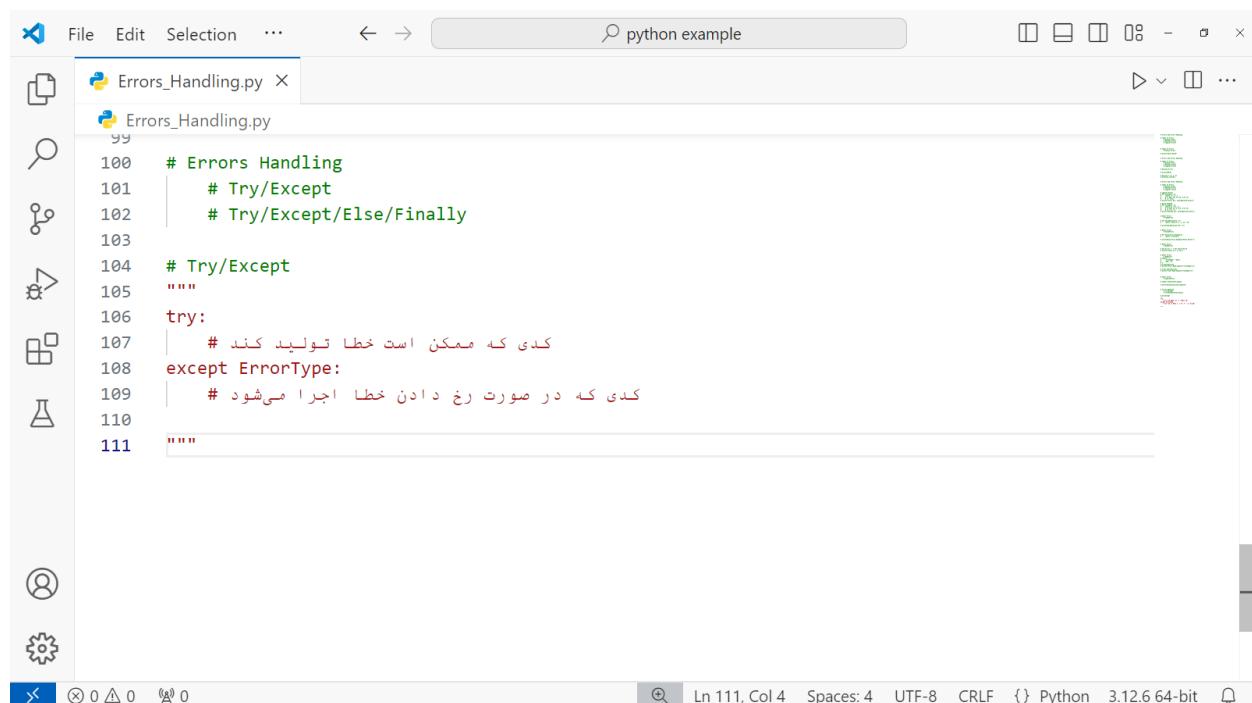
مدیریت خطاهای در زبان برنامه‌نویسی پایتون

۹) مدیریت خطاهای در پایتون

در برنامه‌نویسی، ممکن است خطاهای شرایط غیرمنتظره‌ای رخ دهند که در صورت عدم مدیریت مناسب، باعث متوقف شدن اجرای برنامه شوند. زبان برنامه‌نویسی پایتون ابزار قدرتمندی به نام **مدیریت خطاهای^{۶۶}** ارائه می‌دهد تا این مشکلات را به شیوه‌ای ایمن و ساخت‌یافته کنترل کنید. ساختار اصلی مدیریت استثنایها در پایتون از کلمات کلیدی **try** و **except** استفاده می‌کند.

۱-۹) ساختار برای مدیریت کردن خطاهای **try/except**

ساختار پایه‌ای استفاده از **try** و **except** به شکل زیر است:



```
# Errors Handling
# Try/Except
# Try/Except/Else/Finally

# Try/Except
"""
try:
    # کدی که ممکن است خطا تولید کند
except ErrorType:
    # کدی که در صورت رخ دادن خطا اجرا می‌شود
"""

Ln 111, Col 4 Spaces: 4 UTF-8 CRLF { } Python 3.12.6 64-bit
```

:try بخش •

در این بلوک، کدی قرار می‌گیرد که ممکن است خطایی در حین اجرا تولید کند.

^{۶۶} Exception Handling

• بخش except

اگر خطای در بلوک try رخ دهد، اجرای برنامه متوقف نمی‌شود. در عوض، کنترل به بلوک except منتقل می‌شود و شما می‌توانید پاسخی مناسب برای آن خطای ارائه دهید.

در مثال زیر، برنامه تلاش می‌کند یک عدد را بر صفر تقسیم کند که خطای ZeroDivisionError ایجاد می‌کند:

```
File Edit Selection ... ← → python example
Errors_Handling.py ...
113
114     # Errors Handling
115     |     # Try/Except
116     try:
117         |     result = 265 / 0
118     except ZeroDivisionError:
119         |     print("Division by zero is not possible.")
120

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"
Division by zero is not possible.
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

در اینجا، به جای متوقف شدن برنامه، خطا شناسایی شده و پیام مناسبی نمایش داده می‌شود.

۲-۹) مدیریت چندین نوع خطا

می‌توانید چندین بلوک except برای مدیریت انواع مختلف خطاها تعریف کنید:

The screenshot shows a Python code editor interface. The file being edited is `Errors_Handling.py`. The code demonstrates exception handling:

```
120
121 # Errors Handling
122 | # Try/Except
123 # Many Except Section
124 try:
125     num = int(input("Enter a number:"))
126     result = 10 / num
127 except ValueError:
128     print("Please Enter Number Not Integer!")
129 except ZeroDivisionError:
130     print("Division by zero is not possible!")
```

The terminal window below shows the execution of the script:

```
312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"
Enter a number:hello
Please Enter Number Not Integer!
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python
312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"
Enter a number:0
Division by zero is not possible!
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

۳-۹) استفاده از بلوک عمومی `except`

اگر نوع خطا مشخص نباشد، می‌توانید از بلوک عمومی `except` استفاده کنید:

The screenshot shows a Python code editor interface. The file being edited is `Errors_Handling.py`. The code demonstrates exception handling with a general catch-all block:

```
131
132 # Errors Handling
133 | # Try/Except
134 # If Error is not clear:
135 try:
136     num = int("hello")
137 except:
138     print("An unspecified error has occurred.")
```

The terminal window below shows the execution of the script:

```
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python
312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Errors_Handling.py"
An unspecified error has occurred.
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
```

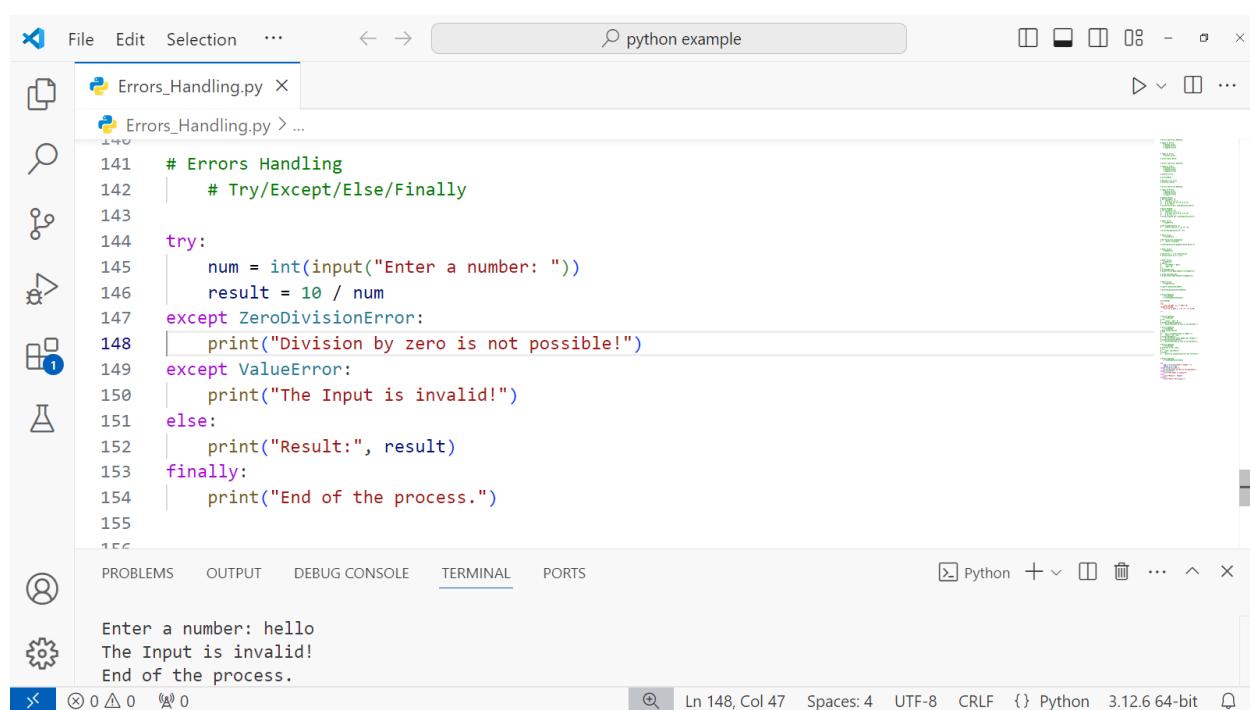
۴-۹) ترکیب با finally و else

برای مدیریت بهتر، می‌توانید از دو بخش else و finally نیز استفاده کنی:

- بخش else: اگر هیچ خطایی در بلوک try رخ ندهد، این بخش اجرا می‌شود.

- بخش finally: این بخش همواره اجرا می‌شود، چه خطا رخ دهد و چه رخ ندهد.

:مثال



```
# Errors Handling
# Try/Except/Else/Finally

try:
    num = int(input("Enter a number: "))
    result = 10 / num
except ZeroDivisionError:
    print("Division by zero is not possible!")
except ValueError:
    print("The Input is invalid!")
else:
    print("Result:", result)
finally:
    print("End of the process.")
```

مقدار ورودی برابر hello

```
File Edit Selection ... ← → python example
Errors_Handling.py X
Errors_Handling.py > ...
140
141 # Errors Handling
142 | # Try/Except/Else/Finally
143
144 try:
145     num = int(input("Enter a number: "))
146     result = 10 / num
147 except ZeroDivisionError:
148     print("Division by zero is not possible!")
149 except ValueError:
150     print("The Input is invalid!")
151 else:
152     print("Result:", result)
153 finally:
154     print("End of the process.")
155
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Enter a number: 0
Division by zero is not possible!
End of the process.
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
Ln 154, Col 33 Spaces: 4 UTF-8 CRLF {} Python 3.12.6 64-bit
```

مقدار ورودی برابر 0

```
File Edit Selection ... ← → python example
Errors_Handling.py X
Errors_Handling.py > ...
140
141 # Errors Handling
142 | # Try/Except/Else/Finally
143
144 try:
145     num = int(input("Enter a number: "))
146     result = 10 / num
147 except ZeroDivisionError:
148     print("Division by zero is not possible!")
149 except ValueError:
150     print("The Input is invalid!")
151 else:
152     print("Result:", result)
153 finally:
154     print("End of the process.")
155
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Enter a number: 20
Result: 0.5
End of the process.
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
Ln 154, Col 33 Spaces: 4 UTF-8 CRLF {} Python 3.12.6 64-bit
```

مقدار ورودی برابر 20

۵-۹) کاربرد مدیریت استثناء

- 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.

مدیریت استثناء یکی از اصول حیاتی برنامه‌نویسی است که به شما اجازه می‌دهد نرم‌افزاری حرفه‌ای‌تر و ایمن‌تر توسعه دهید.

فصل دهم

مفهوم Scope در زبان برنامه‌نویسی پایتون

۱۰) مفهوم دامنه در زبان برنامه‌نویسی پایتون

دامنه^{۶۷} 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. درک دقیق مفهوم دامنه برای نوشتگر کدی ایمن و جلوگیری از مشکلاتی مانند تداخل نام‌ها ضروری است.

۱-۱۰) 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^{۶۸}

۲. دامنه بسته^{۶۹}

۳. دامنه سراسری^{۷۰}

۴. دامنه داخلی^{۷۱}

۱-۱-۱۰) دامنه محلی

متغیرهایی که در داخل یک تابع تعریف می‌شوند و فقط در همان تابع قابل دسترسی هستند.

⁶⁷ Scope

⁶⁸ Local Scope

⁶⁹ Enclosing

⁷⁰ Global Scope

⁷¹ Built-in

```

Scope.py 1 ×
Scope.py > ...
1 # Scope Section
2
3 # Local Scope
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)
12

PROBLEMS 1 OUTPUT DEBUG CONSOLE TERMINAL PORTS
function Result is 45
Traceback (most recent call last):
  File "c:\Users\RezaSeyyed\Desktop\Python\python example\Scope.py", line 9, in <module>
    print(x)
           ^
NameError: name 'x' is not defined
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

Ln 12, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.6 64-bit

```

۲-۱-۱۰) دامنه بسته

متغیرهایی که در داخل یک تابع خارجی (بالادست) تعریف شده‌اند و توسط تابع تو در تو قابل دسترسی هستند.

```

Scope.py ×
Scope.py > ...
13
14 # Scope Section
15 # Enclosing
16 def outer():
17     y = 20 # Enclosing Scope
18     def inner():
19         print(y) # دسترسی به متغیر Enclosing
20     inner()
21 outer()

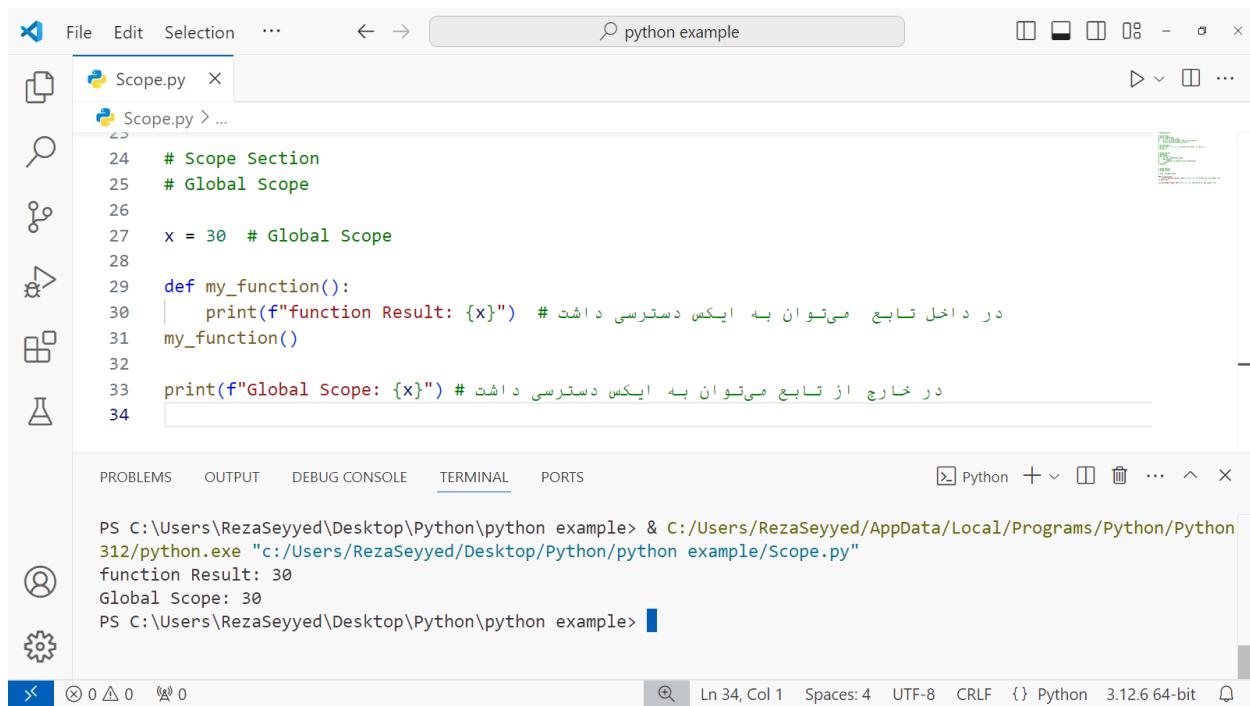
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Scope.py"
20
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

Ln 22, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.6 64-bit

```

۳-۱-۱۰) دامنه سراسری

متغیرهایی که در سطح بالاترین بخش کد تعریف می‌شوند و در کل برنامه قابل دسترسی هستند.



The screenshot shows a VS Code interface with the following details:

- File Bar:** File, Edit, Selection, python example
- Editor:** Scope.py (Content:
24 # Scope Section
25 # Global Scope
26
27 x = 30 # Global Scope
28
29 def my_function():
30 print(f"function Result: {x}")
31 my_function()
32
33 print(f"Global Scope: {x}")
34)
- Output Panel:** PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Scope.py"
function Result: 30
Global Scope: 30
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
- Status Bar:** Ln 34, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.6 64-bit

۴-۱-۱۰) دامنه داخلی

توابع و نامهای از پیش تعریف شده در پایتون که به صورت پیش‌فرض در دسترس هستند، مانند `print()`, `len()` و غیره.

۲-۱۰) 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 = Local).

۲. سپس در دامنه بسته جستجو می‌کند.(E = Enclosing)

۳. اگر پیدا نشد، به دامنه سراسری مراجعه می‌کند(G = Global).

۴. در نهایت، در دامنه داخلی جستجو می‌شود(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 داخل یک تابع به‌طور پیش‌فرض محلی هستند. برای این‌که به این متغیرها را در خارج از تابع دسترسی داشته باشیم باید متغیرهای مورد نظر را از حالت Local به Global تغییر داد و برای این کار، باید از کلیدواژه `global` استفاده کنید.

مثال:

```
# Global Scope and global keyword
# Convert local Scope to global scope

x = 10 # متغیر سراسری
def my_function():
    global x
    x = 20 # تغییر متغیر سراسری
    print(f"In the Function: {x}")
my_function()
print(f"Out the Function: {x}" # مقدار تغییر کرده است
```

PS C:\Users\RezaSeyyed\Desktop\Python\python example> & C:/Users/RezaSeyyed/AppData/Local/Programs/Python/Python 312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Scope.py"
In the Function: 20
Out the Function: 20
PS C:\Users\RezaSeyyed\Desktop\Python\python example>

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.

مثال:

```
Scope.py > ...
48
49 # Local Scope and nonlocal keyword
50 def outer():
51     x = 10 # متغیر Enclosing
52
53     def inner():
54         nonlocal x
55         x = 20 # تغییر مقدار متغیر Enclosing
56         print("Inner:", x)
57
58     inner()
59     print("Outer:", x)
60
61 outer()

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
312/python.exe "c:/Users/RezaSeyyed/Desktop/Python/python example/Scope.py"
Inner: 20
Outer: 20
PS C:\Users\RezaSeyyed\Desktop\Python\python example>
Ln 62, Col 1 Spaces: 4 UTF-8 CRLF {} Python 3.12.6 64-bit
```

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.

درک صحیح مفهوم Scope به شما کمک می‌کند تا از تداخل متغیرها جلوگیری کنید و کدی کارآمد، خوانا و با خطای کمتر بنویسید.

فصل یازدهم

آشنایی با برنامه‌نویسی شی‌گرا

(۱۱) برنامه‌نویسی شی‌گرا

برنامه‌نویسی شی‌گرا^{۷۲} 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^{۷۳} استفاده می‌شود. هر شی، یک موجودیت است که می‌تواند ویژگی‌ها و رفتارهای مخصوص خودش را داشته باشد. برای ساخت این اشیاء از کلاس^{۷۴} 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^{۷۵} و رفتارهای^{۷۶} 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 کیپوله‌سازی^{۷۷}، وراثت^{۷۸} و چندریختی^{۷۹} 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

⁷² Object-Oriented Programming (OOP)

⁷³ Object

⁷⁴ Class

⁷⁵ Attributes

⁷⁶ Methods

⁷⁷ Encapsulation

⁷⁸ Inheritance

⁷⁹ 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 ی داده‌ای در اختیار ندارد، اما از آن می‌توان نمونه‌هایی به نام شی ایجاد کرد.

۱۱-۲) شیء در پایتون

شیء در پایتون یک نمونه^{۸۰} از کلاس است. وقتی یک کلاس را تعریف می‌کنیم، 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.

۱۱-۳) 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.

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 ماشین

مثال ۲. تصور کنید می‌خواهید کلاسی به نام Person تعریف کنید:

➤ کلاس: Person

➤ ویژگی‌ها: نام، سن

⁸⁰ Instance

➤ رفتارها: نمایش اطلاعات

وقتی این کلاس را تعریف کنیم، می‌توانیم شیء‌هایی مانند شخص شماره ۱ و شخص شماره ۲ از آن ایجاد کنیم.

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. سازمان‌دهی 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 شیء 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، بدنه کلاس را کامل می‌کنید.

```
class MyClass:
```

```
x = 5
```

برای تعریف شیء 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) که در داخل کلاس می‌باشد #

__init__() تابع (۷-۱-۱۱)

تابع *(__init__()* در زبان برنامه‌نویسی پایتون یکی از توابع وابسته ویژه^{۸۱} است که به عنوان سازنده^{۸۲} 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.

__init__() نقش تابع (۱-۷-۱-۱۱)

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* است 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 کنیم تا مقادیر اولیه را دریافت و ذخیره کنید.

__init__() ساختار تابع (۲-۷-۱-۱۱)

ساختار کلی این تابع به شکل زیر است:

class ClassName:

^{۸۱} Special Methods

^{۸۲} Constructor

```
def __init__(self, arg1, arg2, ...):  
  
    self.attribute1 = arg1  
  
    self.attribute2 = arg2
```

توضیحات کد:

- Self: اشاره به شیءی که در حال ساخته شدن است.
- attribute1, attribute2: ویژگی هایی هستند که برای شیء تعریف می کنیم و مقدارشان از آرگومان های ورودی گرفته می شود.

۱۱-۱-۷-۳) مثالی از __init__()

در مثال زیر کلاسی به نام Person تعریف شده و از ()__init__ برای مقداردهی اولیه ویژگی های شیء استفاده می شود:

```
class Person:  
  
    def __init__(self, name, age):  
  
        self.name = name # ویژگی name  
  
        self.age = age # ویژگی age  
  
    # ایجاد اشیاء از کلاس Person  
  
person1 = Person("Ali", 25)  
  
person2 = Person("Sara", 30)  
  
# نمایش ویژگی های اشیاء  
  
print(person1.name, person1.age) # خروجی: Ali 25
```

```
print(person2.name, person2.age) # خروجی: Sara 30
```

در این مثال:

- هنگام ساخت `person1` و `person2`, تابع `__init__` به طور خودکار فراخوانی می‌شود و مقادیر `name` و `age` را به ویژگی‌های شیء نسبت می‌دهد.
- ویژگی‌ها با استفاده از `self` به هر شیء نسبت داده می‌شوند.

۴-۷-۱-۱۱ 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 خودمان در زمان ایجاد شیء تنظیم کرد.

۵-۷-۱-۱۱ `self` مفهوم

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.

نکات کلیدی 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.

- برخلاف برخی زبان‌های برنامه‌نویسی دیگر که مرجع نمونه به صورت ضمنی است (مانند "this" در جاوا (Java)، پایتون نیاز دارد تا شما آن را به عنوان اولین پارامتر توابع وابسته نمونه به صراحت 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.

class Person:

```
def __init__(self, name, age):
    self.name = name # Instance attribute
    self.age = age   # Instance attribute

def greet(self):
    return f"Hello, my name is {self.name} and I am {self.age} years old."
```

Creating an object of the class

```
person1 = Person("Alice", 30)
```

Accessing methods and attributes

```
print(person1.greet()) # Output: Hello, my name is Alice and I am 30 years old.
```

self (۱-۷-۶) نحوه عملکرد

- 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 روی همان نمونه فراخوانی شوند.
- نکته: بدون `self`, دسترسی به نمونه را از دست می‌دهید.

به عنوان مثال، اگر `self` را در تعاریف توابع وابسته حذف کنید، پایتون خطای خواهد داد:

`class Person:`

```
def __init__(name, age): # Missing 'self'

    self.name = name    # Error: 'self' is not defined

    self.age = age

def greet():

    return f"Hello, my name is {self.name} and I am {self.age} years old."
```

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.

۸-۱-۱۱ `__str__()` تابع

تابع `'__str__()'` یکی از توابع وابسته‌ی جادویی (magic methods) در پایتون است که برای تعریف نحوه نمایش "خوانای کاربر" (user-friendly representation) 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 فراخوانی می‌شود. این تابع معمولاً برای بازگشت یک رشته خوانا و توصیفی از شیء استفاده می‌شود که می‌تواند برای کاربران مناسب باشد.

مثال:

```
```python
class Person:

 def __init__(self, name, age):
 self.name = name
 self.age = age

 def __str__(self):
 return f"Name: {self.name}, Age: {self.age}"

ساخت یک شیء
person = Person("Ali", 25)

استفاده از print یا str
print(person) # خروجی: Name: Ali, Age: 25
print(str(person)) # خروجی: Name: Ali, Age: 25
```

```

۹-۱-۱۱ `__repr__()` تابع

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 ادامه به کاربرد تابع `().__repr__()` پرداخته می‌شود.

۱. نمایش رسمی(**Developer-Friendly**): وقتی از تابع `repr()` روی یک شیء استفاده می‌کنید یا در

محیط‌هایی مثل ترمینال پایتون، شیء را تایپ می‌کنید، این تابع فراخوانی می‌شود.

۲. 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).

مثال:

class Person:

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

```
    self.name = name
```

```
    self.age = age
```

```
def __repr__(self):
```

```
    return f"Person(name={self.name!r}, age={self.age!r})"
```

```
# ساخت یک شیء
```

```
person = Person("Ali", 25)
```

```
# استفاده از repr
```

```
print(repr(person)) # خروجی: Person(name='Ali', age=25)
```

```
# در کسول پایتون #  
  
# >>> person  
  
# 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، شیء به صورت زیر نمایش داده می‌شود:

`<*ClassName* object at *memory_address*>`

```
'''python  
  
class Person:  
  
    def __init__(self, name, age):  
  
        self.name = name  
  
        self.age = age  
  
    person = Person("Ali", 25)  
  
    print(person) # <__main__.Person object at 0x7f2b6c3f6700>  
  
'''
```

۱۱-۱۰-۱۱) تفاوت بین __str__() و __repr__()

__str__(): نمایش رشته‌ای، مناسب برای کاربران. باید "خوانا و دوستانه" باشد.
__repr__(): نمایش رشته‌ای، مناسب برای توسعه‌دهندگان. باید شامل اطلاعات کافی باشد تا بتوان با آن، شیء را بازسازی کرد.

مثال مقایسه‌ای:

```
```python
class Person:

 def __init__(self, name, age):
 self.name = name
 self.age = age

 def __str__(self):
 return f"Name: {self.name}, Age: {self.age}"

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

person = Person("Ali", 25)

print(person) # خروجی از __str__: Name: Ali, Age: 25

print(repr(person)) # خروجی از __repr__: 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()` یا `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.

انواع توابع وابسته در کلاس‌های پایتون

۱. توابع وابسته‌ی نمونه (Instance Methods)

- به داده‌ها و ویژگی‌های یک نمونه خاص از کلاس دسترسی دارند.
- اولین آرگومان آن‌ها باید `self` باشد که به نمونه جاری اشاره می‌کند.

۲. توابع وابسته‌ی کلاس (Class Methods)

- 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 (Static Methods)

- 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.

۱۱-۱-۲) تعریف و مثال برای هر نوع تابع وابسته

۱. تابع وابسته نمونه (Instance Method)

```
class Person:  
  
    def __init__(self, name, age):  
  
        self.name = name # ویژگی نمونه  
  
        self.age = age  
  
    def greet(self):  
  
        return f"Hello, my name is {self.name} and I am {self.age} years old."  
  
# استفاده از تابع وابسته نمونه  
  
person1 = Person("Ali", 25)  
  
print(person1.greet()) # خروجی: Hello, my name is Ali and I am 25 years old.
```

۲. تابع وابسته کلاس (Class Method)

```
class Person:  
  
    population = 0 # ویژگی کلاس  
  
    def __init__(self, name):  
  
        self.name = name  
  
        Person.population += 1  
  
    @classmethod  
  
    def get_population(cls):  
  
        return f"There are {cls.population} people."  
  
# استفاده از تابع وابسته کلاس
```

```
print(Person.get_population()) # خروجی: There are 0 people.
```

```
person1 = Person("Ali")
```

```
print(Person.get_population()) # خروجی: There are 1 people.
```

۳. تابع وابسته ایستا (Static Method)

```
class MathUtils:
```

```
    @staticmethod
```

```
    def add(a, b):
```

```
        return a + b
```

```
استفاده از تابع وابسته ایستا #
```

```
print(MathUtils.add(5, 3)) # خروجی: 8
```

نکات:

۱. تابع وابسته‌ی نمونه:

- برای کار با داده‌های مرتبط با یک نمونه خاص از کلاس استفاده می‌شود.

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.

تعریف چند تابع وابسته‌ی مختلف در یک کلاس

class Calculator:

```
def __init__(self, name):  
    self.name = name  
  
def instance_method(self, x, y):  
    return f"{self.name} adds: {x + y}"
```

@classmethod

```
def class_method(cls, x, y):  
    return f"Class adds: {x + y}"
```

@staticmethod

```
def static_method(x, y):  
    return f"Static adds: {x + y}"
```

استفاده از توابع وابسته #

```
calc = Calculator("MyCalculator")  
  
print(calc.instance_method(3, 4)) # خروجی: MyCalculator adds: 7  
  
print(Calculator.class_method(3, 4)) # خروجی: Class adds: 7  
  
print(Calculator.static_method(3, 4)) # خروجی: Static adds: 7
```

نکات:

- از تابع وابسته‌ی نمونه برای دسترسی و تغییر ویژگی‌های یک شیء استفاده می‌کنیم.
- از تابع وابسته‌ی کلاس برای دسترسی و تغییر ویژگی‌های کلاس استفاده می‌کنیم.
- از تابع وابسته‌ی ایستا برای عملیات‌های مستقل که به داده‌های کلاس یا نمونه نیاز ندارند استفاده می‌کنیم.

۲-۱۱) 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.

۱-۲-۱۱) ویژگی‌های کلیدی وراثت

- **کلاس والد (کلاس پایه):** کلاسی که از آن ارثبری می‌شود.
- **کلاس فرزند (کلاس مشتق شده):** کلاسی که از کلاس والد ارثبری 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 ارثبری کند.

۲-۲-۱۱) نحوه استفاده از وراثت در پایتون

class ParentClass:

```
# Parent class definition
```

```
class ChildClass(ParentClass):
```

```
    # Child class definition
```

۱۱-۲-۳) مثالی از وراثت تک‌گانه

```
class Animal:
```

```
    def speak(self):
```

```
        return "I make a sound."
```

```
class Dog(Animal):
```

```
    def speak(self):
```

```
        return "Woof! Woof!"
```

```
# Create objects
```

```
animal = Animal()
```

```
dog = Dog()
```

```
print(animal.speak()) # Output: I make a sound.
```

```
print(dog.speak()) # Output: Woof! Woof!
```

نکته: کلاس های فرزند می‌توانند توابع وابسته‌ی کلاس والد را بازنویسی کنند.

```
class Parent:
```

```
    def greet(self):
```

```
        return "Hello from the parent class!"
```

```

class Child(Parent):

    def greet(self):
        return "Hello from the child class!"

# Test overriding

obj = Child()

print(obj.greet()) # Output: 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 (فرزنده) 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 موقعيت کاربرد دارد که توابع وابسته والد باز تعریف شده‌اند (Method Overriding).

یکی از اهداف اصلی استفاده از `super()`، ساده کردن مدیریت ارثبری در کلاس‌ها، خصوصاً در ساختارهای پیچیده مانند ارثبری چندگانه و چندسطحی است. این تابع از مکانیزم ترتیب جستجوی تابع وابسته - (MRO) 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 ارثبری 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 ارثبری 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.

مثالی از `super()`

```
class Parent:
```

```
    def greet(self):  
        return "Hello from Parent!"
```

```
class Child(Parent):
```

```
    def greet(self):  
        return super().greet() + " And hello from Child!"
```

```
# Test super()
```

```
obj = Child()  
  
print(obj.greet()) # Output: Hello from Parent! And hello from Child!
```

۱۱-۲-۵) مثالی از وراثت چندگانه

پایتون از وراثت چندگانه پشتیبانی می‌کند و به یک کلاس اجازه می‌دهد از چندین کلاس والد ارثبری کند.

```
class A:
```

```
    def method_a(self):  
        return "Method from class A"
```

```
class B:
```

```

def method_b(self):
    return "Method from class B"

class C(A, B):
    pass

# Test multiple inheritance
obj = C()
print(obj.method_a()) # Output: Method from class A
print(obj.method_b()) # Output: Method from class B

```

۱۱-۲-۶) مثالی از وراثت چندسطحی

نکته: یک کلاس فرزند می‌تواند از کلاس فرزند دیگر ارث برد.

```

class Animal:
    def breathe(self):
        return "I am breathing"

```

```

class Mammal(Animal):
    def walk(self):
        return "I am walking"

```

```

class Dog(Mammal):
    def bark(self):

```

```

return "Woof! Woof!"

# Test multi-level inheritance

dog = Dog()

print(dog.breathe()) # Output: I am breathing

print(dog.walk())   # Output: I am walking

print(dog.bark())  # Output: Woof! Woof!

```

۱۱-۳) چندریختی در پایتون

چندریختی در پایتون یک مفهوم برنامه‌نویسی شی‌ءگرا است که اجازه می‌دهد از یک رابط یکسان (تابع وابسته) برای اشیاء از انواع مختلف استفاده شود. این امر رویکردی یکپارچه و انعطاف‌پذیر برای نوشتن کد را ممکن می‌سازد، زیرا می‌توانید از یک تابع یا تابع وابسته یکسان در کلاس‌های مختلف استفاده کنید، به شرطی که همان رابط را پیاده‌سازی کنند.

چندریختی در برنامه‌نویسی شی‌ءگرا به این مفهوم اشاره دارد که یک تابع وابسته، یا شیء می‌تواند رفتارهای مختلفی داشته باشد، بسته به نوع داده‌ها یا کلاس‌هایی که با آن‌ها کار می‌کند. در پایتون، این ویژگی به دلیل پویایی و تایپ‌گذاری ضعیف زبان بسیار پررنگ است و انعطاف زیادی در کدنویسی فراهم می‌کند.

چندریختی به ما اجازه می‌دهد که از یک رابط یا یک تابع وابسته مشترک برای اشیای مختلف استفاده کنیم، بدون اینکه نگران نوع یا کلاس آن اشیا باشیم. به عبارت ساده‌تر، اگر دو یا چند کلاس توابع وابسته با نام یکسان داشته باشند، می‌توان با استفاده از همان تابع وابسته، رفتارهای متفاوتی در اشیای مختلف مشاهده کرد. این ویژگی باعث می‌شود که ساده‌تر و انعطاف‌پذیرتر باشد.

۱-۳-۱۱) انواع چندریختی در پایتون

۱. چندریختی مبتنی بر ارثبری: در این حالت، کلاس‌های فرزند توابع وابسته کلاس والد را بازنویسی

می‌کنند. بنابراین، تابع وابسته مشترک در کلاس‌های 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) مثال‌های مربوط به مفهوم چندریختی

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 یکی از اصول چندریختی است.

class Animal:

```

def sound(self):
    return "Some generic sound"

class Dog(Animal):
    def sound(self):
        return "Woof! Woof!"

class Cat(Animal):
    def sound(self):
        return "Meow!"

animals = [Dog(), Cat(), Animal()]

for animal in animals:
    print(animal.sound())

```

Duck Typing (۲-۳-۳-۱۱)

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.

```

class Bird:
    def fly(self):
        return "Flying high"

class Airplane:
    def fly(self):
        return "Taking off"

def lift_off(entity):
    print(entity.fly())

```

```
lift_off(Bird())    # Output: Flying high  
lift_off(Airplane()) # Output: Taking off
```

نکته: اگرچه چندريختی اغلب با ارثبری همراه است، اما اين دو مفهوم يكسان نیستند. 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) مازول‌ها در پایتون

مازول‌ها در پایتون فایل‌هایی هستند که شامل کد پایتون (مثل توابع، کلاس‌ها، متغیرها و غیره) می‌باشند و می‌توان آن‌ها را در برنامه‌های دیگر وارد (import) کرده و دوباره استفاده کرد. مازول‌ها به سازمان‌دهی منطقی کد، افزایش قابلیت استفاده مجدد و ساده‌تر شدن نگهداری کمک می‌کنند. این مازول‌ها می‌توانند داخلی (از پیش فراهم‌شده توسط پایتون) یا تعریف‌شده توسط کاربر باشند.

۱-۱۲) چرا باید از مازول‌ها استفاده کنیم؟

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 مازول جداگانه قرار دهید.

۲-۱۲) انواع مازول‌ها

۱-۲-۱۲) مازول‌های داخلی

این مازول‌ها از پیش در پایتون نصب شده‌اند. مثال‌ها شامل random، math، os، sys، و

مثال:

```
import math
```

```
print(math.sqrt(16)) # ۴.۰
```

۲-۲-۱۲) ماثول‌های تعریف شده توسط کاربر

ماژول‌هایی که توسط کاربران برای استفاده مجدد از کد سفارشی ایجاد شده است.

```
# my_module.py

def greet(name):

    return f'Hello, {name}!'

# main.py

import my_module

print(my_module.greet("Alice"))
```

۳-۲-۱۲) ماثول‌های شخص ثالث

ماژول‌هایی که توسط خود شرکت سازنده‌ی پایتون ارائه شده است و از طریق `pip` نصب می‌شوند مانند `numpy`،
... و `pandas`

```
pip install numpy

import numpy as np

print(np.array([1, 2, 3]))
```

۳-۱۲) وارد کردن ماژول‌ها به داخل پروژه

برای وارد کردن ماژولی 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 ماژول را بنویسید.

import module_name

۲-۳-۱۲) وارد کردن مقدار خاص مانند کلاس از یک ماژول

در این روش، در مرحله اول کلیدواژه `from` را نوشت، در مرحله‌ی دوم نام ماژول، در مرحله سوم کلیدواژه `import` و در مرحله چهارم نام کلاس‌ها یا مقادیری که در ماژول وجود دارد، نوشت، می‌شود.

`from module_name import member1, member2`

۳-۳-۱۲) وارد کردن تمامی مقادیری که در داخل ماژول 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 *`*

۴-۳-۱۲) وارد کردن ماژول با دادن نام مستعار برای آن ماژول

در این روش، در مرحله اول کلیدواژه `import` را نوشت و در مرحله دوم نام ماژول، 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.

۵-۳-۱۲) مثال‌های مربوط به وارد کردن مازول‌ها

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 از کدهای زیر به عنوان نمونه قرار داده شده است.

```
def add(a, b):
```

```
    return a + b
```

```
def subtract(a, b):
```

```
    return a - b
```

۲) یک فایل دیگر پایتون در همان فolder در کنار فایل پایتون که در مرحله قبل ایجاد شد، ایجاد کرده و برای این که بتوان از فایل پایتون مرحله قبل به عنوان ماثول در این فایل پایتون استفاده کنید نیازه که ماثول را وارد کنید.

```
import my_module
```

سپس برای استفاده از توابع فایل `my_module.py` طبق روال زیر عمل کنید.

```
print(my_module.add(5, 3))    # Output: 8
```

```
print(my_module.subtract(5, 3)) # Output: 2
```

۵-۱۲) متغیر `__name__` در پایتون

متغیر `__name__` یک متغیر داخلی و خاص در پایتون است که نشان‌دهنده‌ی نام یک ماثول است. این متغیر به صورت خودکار توسط مفسر پایتون مقداردهی می‌شود و در تعیین اینکه یک ماثول مستقیماً اجرا شده یا به عنوان ماثول 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 "`"__main__"`" خواهد بود.

۲. اگر فایل پایتون به عنوان یک ماثول در برنامه دیگری وارد شود، مقدار `__name__` برابر با نام آن ماثول 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 ماثول‌ها ضروری است.

۱۲-۵-۳) مثال‌های مربوط به `__name__`

مثال ۱. فرض کنید یک فایل پایتون به نام `test.py` داریم:

```
# test.py

def greet():

    print("Hello, world!")

if __name__ == "__main__":
    print("This file is executed directly.")

    greet()
```

حالتهای اجرا:

۱. اجرای مستقیم فایل: اگر فایل را مستقیماً اجرا کنید:

خروجی:

This file is executed directly.

Hello, world!

۲. وارد کردن فایل به عنوان ماثول: اگر فایل را در یک برنامه دیگر وارد کنید:

```
import test
```

خروجی:

هیچ‌کدام از کدهای داخل `if __name__ == "__main__":` اجرا نمی‌شوند.

مثال ۲. فرض کنید دو فایل دارید:

module1.py ✓

```
def add(a, b):  
  
    return a + b  
  
if __name__ == "__main__":  
  
    print("module1 is executed directly.")  
  
    print(add(5, 3))
```

main.py ✓

```
import module1  
  
print("main.py is running.")  
  
result = module1.add(10, 20)  
  
print("Result:", result)
```

نتایج اجرا:

module1.py ✦ اجرای

خروجی:

module1 is executed directly.

8

main.py ✦ اجرای

خروجی:

main.py is running.

Result: 30

در این مثال، کد داخل `if __name__ == "__main__":` در فایل `module1.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 الگوریتم در شبیه‌سازی دارد و اعداد تولیدی آن به خوبی از نظر آماری "تصادفی" رفتار می‌کنند.

۳. کاربردهای گسترده:

- این مژول 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 لیست‌ها 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 `random`

۱. شبه‌تصادفی بودن: این مژول 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: مژول `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 `namelist`) ترجیح داده شود.

نکات:

- `Random` بخشی از کتابخانه استاندارد پایتون است و نیازی به نصب جداگانه ندارد.
- این مژول 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 کاربردهای امنیتی) در دسترس هستند.

۳-۱۳) توابع وابسته در ماثول random

در جدول تمامی توابع وابسته موجود در ماثول **random** و توضیح کوتاه برای هر تابع وابسته آمده است:

| 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 (لیست، تاپل، یا رشته). | choice(seq) |
| انتخاب k عنصر تصادفی از دنباله با امکان وزن دهنده. | choices(population, weights=None, k=1) |
| انتخاب تصادفی k عنصر بدون جای‌گذاری از دنباله. | sample(population, k) |
| مرتب‌سازی تصادفی عناصر لیست (تغییر در جای خود). | shuffle(seq) |
| تنظیم دانه (Seed) برای تکرارپذیری اعداد تصادفی. | seed(a=None, version=2) |
| دربافت 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 (شبه‌نرمال). | normalvariate(mu, sigma) |
| تولید عدد تصادفی با توزیع ون میزس (برای داده‌های زاویه‌ای). | vonmisesvariate(mu, kappa) |
| تولید عدد تصادفی با توزیع پارتو. | 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 `random`

۱-۴-۱۳) تولید اعداد تصادفی

۱-۱-۴-۱۳) تولید عددی تصادفی بین بازه $[0,1]$ (شامل صفر و بدون یک).

`random.random()`

`print(random.random())`

۲-۱-۴-۱۳) تولید عددی اعشاری تصادفی بین a و b .

`random.uniform(a, b)`

`print(random.uniform(5, 10))`

۲-۴-۱۳) تولید اعداد صحیح

۱۳-۴-۲-۱) تولید عدد صحیح تصادفی بین a و b (شامل هر دو).

random.randint(a, b)

```
print(random.randint(1, 10))
```

۱۳-۴-۲-۲) تولید عدد صحیح تصادفی از دنباله‌ای با فاصله مشخص.

random.randrange(start, stop[, step])

```
print(random.randrange(0, 10, 2))
```

۱۳-۴-۳) کار با لیست‌ها

۱۳-۴-۳-۱) انتخاب تصادفی یک عنصر از دنباله (لیست، تاپل یا رشته).

random.choice(seq)

```
fruits = ['apple', 'banana', 'cherry']
```

```
print(random.choice(fruits))
```

۱۳-۴-۳-۲) انتخاب تصادفی k عنصر از دنباله با امکان تعریف وزن.

random.choices(seq, weights=None, k=1)

```
fruits = ['apple', 'banana', 'cherry']

print(random.choices(fruits, weights=[1, 3, 1], k=5))
```

. ۱۳-۴-۳-۳) چیدمان تصادفی عناصر یک لیست (در جای خود تغییر می‌دهد.

random.shuffle(seq)

```
fruits = ['apple', 'banana', 'cherry']

random.shuffle(fruits)

print(fruits)
```

. ۱۳-۴-۳-۴) نمونه‌گیری از یک لیست، انتخاب k عنصر تصادفی بدون جای‌گذاری از دنباله.

random.sample(seq, k)

```
fruits = ['apple', 'banana', 'cherry']

print(random.sample(fruits, 2))
```

. ۱۳-۴-۴) توابع توزیع آماری

. ۱۳-۴-۴-۱) تولید عدد تصادفی با توزیع نرمال (گاوسی).

random.gauss(mu, sigma)

```
print(random.gauss(0, 1))
```

۱۳-۴-۴-۲) تولید عدد تصادفی با توزیع نمایی.

random.expovariate(lambd)

```
print(random.expovariate(1.5))
```

۱۳-۴-۴-۳) تولید عدد تصادفی با توزیع بتا.

random.betavariate(alpha, beta)

```
print(random.betavariate(2, 5))
```

نکته: تعیین دانه (Seed)

برای تولید اعداد تصادفی تکرارپذیر (برای مثال در آزمایشات)، از `random.seed()` استفاده کنید.

```
random.seed(42)
```

```
print(random.random()) # خروجی همواره ثابت خواهد بود
```

فصل چهاردهم

آشنایی با ماثول Math در پایتون

۱۴) آشنایی با ماژول 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. **توابع عمومی ریاضی:** این ماژول شامل توابع پایه‌ای مانند جمع، تفریق، ضرب و تقسیم نیست (زیرا این عملیات‌ها مستقیماً در پایتون پشتیبانی می‌شوند)، اما توابع 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. **توابع مثلثاتی و زاویه‌ای:** توابع 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. **توابع لگاریتمی و نمایی:** ماژول **math** ابزارهایی برای محاسبه لگاریتم در پایه‌های مختلف (مانند پایه ۲ یا پایه ۱۰) و همچنین توابع نمایی برای توان‌های طبیعی یا عدد نپر ارائه می‌دهد. این توابع 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 پی و عدد نپر است که به دقت بالا در محاسبات علمی کمک می‌کند. همچنین، ثابت‌های دیگری مانند تاو، که دو برابر عدد پی است.

۵. توابع آماری و اعداد تصادفی: گرچه **math** به طور مستقیم برای تولید اعداد تصادفی طراحی نشده است این کار توسط ماژول **random** انجام می‌شود، توابع آماری مانند میانگین هندسی و ابزارهای مرتبط با اعداد اول و ترکیبات در این ماژول قرار دارند

۶. ماژول **math** به طور گسترده از توابع ریاضی استاندارد کتابخانه **C (libm)** استفاده می‌کند. این بدان معناست که بسیاری از توابع موجود در این ماژول، مستقیماً با استفاده از کتابخانه ریاضی زبان C پیاده‌سازی شده‌اند. این طراحی دو مزیت کلیدی دارد:

➤ سرعت بالا: توابع این ماژول، 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.

۱۴-۲) کاربردهای ماژول **Math**

ماژول **math** برای برنامه‌هایی که نیاز به دقت بالا در محاسبات ریاضی دارند، ایده‌آل است. این ماژول 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 ماژول 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.
- مدل‌سازی ریاضی و حل معادلات شامل توزیع‌های خاص، توابع لگاریتمی، و توابع مثلثاتی معکوس.

- طراحی سیستم‌های کنترل، شبیه‌سازی‌های عددی و تحلیل داده‌ها.

۲. گرافیک کامپیوتروی و بازی‌سازی

- استفاده از توابع مثلثاتی برای محاسبات مرتبط با چرخش، مقیاس‌دهی و حرکت اشیاء در فضای سه‌بعدی.
- محاسبه فاصله بین نقاط، بردارها، یا زوایا در بازی‌ها یا شبیه‌سازی‌های گرافیکی.
- تنظیمات دوربین و نورپردازی در گرافیک سه‌بعدی با استفاده از توابع مثلثاتی.

۳. علوم داده و تحلیل آماری

- پردازش داده‌ها و استفاده از توابع لگاریتمی و نمایی برای تغییر مقیاس داده‌ها.
- محاسبه مقادیر میانگین هندسی و استفاده از توابع توزیع خاص برای شبیه‌سازی داده‌ها.
- استفاده از ثابت‌های ریاضی مانند π برای تحلیل‌های هندسی.

۴. شبیه‌سازی‌های علمی

- محاسبات مرتبط با موج‌ها، ارتعاشات یا نوسانات با استفاده از توابع مثلثاتی و نمایی.
- شبیه‌سازی رفتار سیستم‌های دینامیکی با توابع مرتبط با لگاریتم یا توزیع‌های خاص.

۵. هوش مصنوعی و یادگیری ماشین

- استفاده در پیش‌پردازش داده‌ها مانند نرمال‌سازی با توابع \log یا محاسبات توان.
- پیاده‌سازی الگوریتم‌های مبتنی بر هندسه یا بهینه‌سازی.
- محاسبات فاصله (مانند فاصله اقلیدسی) برای الگوریتم‌های دسته‌بندی یا خوشه‌بندی.

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 فرآیندهای تکاملی.

• استفاده در مدل‌سازی‌های زیست‌محیطی یا تغییرات اقلیمی.

۱۴-۳) توابع وابسته موجود در ماژول math

در جدول، تمامی توابع وابسته‌ی ماژول math همراه با توضیح آن‌ها آورده شده است:

| تابع وابسته | توضیحات |
|----------------------------|---|
| ceil(x) | مقدار عدد x را به نزدیک‌ترین عدد صحیح بزرگ‌تر گرد می‌کند. |
| floor(x) | مقدار عدد x را به نزدیک‌ترین عدد صحیح کوچک‌تر گرد می‌کند. |
| fabs(x) | مقدار مطلق عدد x را برمی‌گردد (بدون در نظر گرفتن علامت). |
| factorial(x) | فاکتوریل عدد صحیح x را محاسبه می‌کند. |
| fsum(iterable) | مجموع دقیق عناصر یک iterable با دقت بالا را برمی‌گردد. |
| gcd(a, b) | بزرگ‌ترین مقسوم‌علیه مشترک (ب.م.م) دو عدد صحیح a و b را برمی‌گردد. |
| isqrt(n) | جذر عدد صحیح n را به صورت عدد صحیح برمی‌گردد. |
| sqrt(x) | جذر عدد x را محاسبه می‌کند. |
| exp(x) | مقدار e^x (عدد نپیر به توان x) را برمی‌گردد. |
| expm1(x) | مقدار $1 - e^x$ را با دقت بالا برای مقادیر کوچک x محاسبه می‌کند. |
| log(x[, base]) | لگاریتم عدد x را در پایه مشخص شده (پیش‌فرض: پایه e) محاسبه می‌کند. |
| log10(x) | لگاریتم عدد x در پایه ۱۰ را محاسبه می‌کند. |
| log2(x) | لگاریتم عدد x در پایه ۲ را محاسبه می‌کند. |
| pow(x, y) | مقدار $y^{**}x$ را برمی‌گردد. |
| prod(iterable, *, start=1) | حاصل ضرب عناصر یک iterable را محاسبه می‌کند. |
| modf(x) | قسمت اعشاری و صحیح عدد x را به صورت دوتایی بازمی‌گردد. |
| trunc(x) | بخش صحیح عدد x را با حذف قسمت اعشاری برمی‌گردد. |
| sin(x) | مقدار سینوس زاویه x (بر حسب رادیان) را محاسبه می‌کند. |
| cos(x) | مقدار کسینوس زاویه x (بر حسب رادیان) را محاسبه می‌کند. |
| tan(x) | مقدار تانژانت زاویه x (بر حسب رادیان) را محاسبه می‌کند. |

| | |
|---|-------------------------------------|
| سینوس معکوس عدد x را بر حسب رادیان برمی‌گرداند. | $\text{asin}(x)$ |
| کسینوس معکوس عدد x را بر حسب رادیان برمی‌گرداند. | $\text{acos}(x)$ |
| تانژانت معکوس عدد x را بر حسب رادیان برمی‌گرداند. | $\text{atan}(x)$ |
| زاویه قطبی (آرکتانژانت) برای مختصات (y, x) را برمی‌گرداند. | $\text{atan2}(y, x)$ |
| مقدار زاویه x را از رادیان به درجه تبدیل می‌کند. | $\text{degrees}(x)$ |
| مقدار زاویه x را از درجه به رادیان تبدیل می‌کند. | $\text{radians}(x)$ |
| طول هیپوتنوس (وتر) برای مجموعه‌ای از مختصات را محاسبه می‌کند. | $\text{hypot}(*\text{coordinates})$ |
| تابع گاما برای عدد x را محاسبه می‌کند. | $\text{gamma}(x)$ |
| لگاریتم طبیعی تابع گاما برای عدد x را محاسبه می‌کند. | $\text{lgamma}(x)$ |
| مقدار تابع خطای گاوسی برای عدد x را برمی‌گرداند. | $\text{erf}(x)$ |
| مقدار تابع خطای مکمل گاوسی برای عدد x را برمی‌گرداند. | $\text{erfc}(x)$ |
| مقدار عدد پی را ذخیره می‌کند. | pi |
| مقدار عدد نپر (e) را ذخیره می‌کند. | e |
| مقدار عدد تاو را ذخیره می‌کند. | tau |
| مقدار بی‌نهایت مثبت را ذخیره می‌کند. | inf |
| مقدار Not a Number (NaN) را ذخیره می‌کند. | nan |

۴-۱۴) توابع مهم و پرکاربرد مازول math

۱-۴-۱۴) محاسبات عددی

۱-۱-۴-۱۴) تابع وابسته‌ی محاسبه رادیکال یک عدد.

$\text{math.sqrt}(x)$

مثال:

خروجی:

۱۴-۴-۲) تابع وابسته‌ی محاسبه توان یک عدد.

math.pow(x, y)

مثال:

خروجی:

۱۴-۴-۳) تابع وابسته‌ی محاسبه فاکتوریل یک عدد صحیح.

math.factorial(x)

مثال:

خروجی:

۱۴-۱-۴) تابع وابسته‌ی محاسبه مجموع دقیق عناصر یک فهرست.

math.fsum(iterable)

مثال:

خروجی:

۱۴-۲) توابع مثلثاتی.

۱۴-۳-۱) توابع وابسته‌ی محاسبه سینوس، کسینوس و تانژانت.

math.sin(x)

math.cos(x)

math.tan(x)

مثال:

خروجی:

۱۴-۴-۲-۲) تابع وابسته‌ی محاسبه سینوس معکوس، کسینوس معکوس و تانژانت معکوس.

math.asin(x)

math.acos(x)

math.atan(x)

مثال:

خروجی:

۱۴-۴-۲-۳) تابع وابسته‌ی تبدیل رادیان به درجه.

math.degrees(x)

مثال:

خروجی:

۱۴-۴-۲-۴) تابع وابسته‌ی تبدیل درجه به رادیان.

math.radians(x)

مثال:

خروجی:

۱۴-۴-۳) لگاریتم‌ها و توابع نمایی

۱۴-۴-۳-۱) تابع وابسته‌ی محاسبه توابع نمایی.

math.exp(x)

مثال:

خروجی:

۲-۳-۴-۱۴) تابع وابسته‌ی محاسبه لگاریتم یک عدد.

math.log(x[, base])

مثال:

خروجی:

۳-۳-۴-۱۴) تابع وابسته‌ی محاسبه لگاریتم یک عدد بر پایه‌ی ۱۰.

math.log10(x)

مثال:

خروجی:

۴-۳-۴-۱۴) تابع وابسته‌ی محاسبه لگاریتم یک عدد بر پایه‌ی ۲.

Math.log2(x)

مثال:

خروجی:

۱۴-۴-۴-۲) توابع ویژه

۱۴-۴-۴-۱) تابع وابسته‌ی گاما.

math.gamma(x)

مثال:

خروجی:

۱۴-۴-۴-۲) تابع وابسته‌ی محاسبه لگاریتم طبیعی تابع گاما.

math.lgamma(x)

مثال:

خروجی:

۱۴-۴-۴-۳) تابع وابسته‌ی محاسبه مقدار تابع خطای گاوسی برای عدد.

math.erf(x)

مثال:

خروجی:

۱۴-۴-۴-۴) تابع وابسته‌ی محاسبه مقدار تابع خطای مکمل گاوسی.

math.erfc(x)

مثال:

خروجی:

۱۴-۴-۵) مدیریت اعداد اعشاری و صحیح

۱۴-۴-۵-۱) تابع وابسته‌ی گرد کردن عددی به سمت عدد بزرگ‌تر.

math.ceil(x)

مثال:

خروجی:

۱۴-۴-۵-۲) تابع وابسته‌ی گرد کردن عددی به سمت عدد کوچک‌تر.

math.floor(x)

مثال:

خروجی:

۱۴-۴-۵-۳) تابع وابسته حذف قسمت اعشاری یک عدد اعشاری.

math.trunc(x)

مثال:

خروجی:

۱۴-۴-۵-۴) تابع وابسته جداسازی قسمت صحیح و اعشاری یک عدد اعشاری.

math.modf(x)

مثال:

خروجی:

۱۴-۴-۶) ثابت‌های ریاضی

۱۴-۴-۶) عدد پی

math.pi

مثال:

خروجی:

۱۴-۴-۶) عدد نپر

math.e

مثال:

خروجی:

(۳-۶-۴-۱۴) عدد تاو (دو برابر مقدار عدد پی)

math.tau

مثال:

خروجی:

(۴-۶-۴-۱۴) مقدار بی‌نهایت

math.inf

مثال:

خروجی:

(۵-۶-۴-۱۴) مقدار NaN یا به اصطلاح Not a Number

math.nan

مثال:

خروجی:

۱۴-۵) 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 لگاریتم‌ها 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

Datetime آشنایی با ماژول (۱۵)

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 ماژول‌های 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 متعدد 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` می‌باشد.

۲. ایجاد و مدیریت تاریخ‌ها و زمان‌ها: کلاس‌ها به شما امکان می‌دهند تا بخش‌های مختلف تاریخ و زمان را مدیریت کنید.

۳. فرمتسازی و تبدیل تاریخ‌ها

- 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.

۴. توانایی تبدیل تاریخ‌ها و زمان‌ها به فرمتهای مختلف مانند متن، عدد، و فرمتهای استاندارد 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. تایم‌زونی 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 معتمد: 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 نادرست مانند `.infinity` و `NaT` (Not a Time)
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.

۱۵-۲) کاربردهای مژول **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 لاغ‌گیری 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. اتماسیون‌ها 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.

۱۵-۳) توابع وابستهٔ موجود در مژول `Datetime`

ماژول `datetime` در پایتون شامل کlassen‌ها و متدهایی برای کار با تاریخ و زمان است. در جدول زیر، متدهای مهم این ماژول و توضیحات آن‌ها آمده است:

| توضیحات | تابع وابسته |
|---|--------------------------|
| تاریخ و زمان فعلی را بر اساس زمان سیستم برمی‌گرداند. | datetime.now() |
| تاریخ و زمان امروز را برمی‌گرداند (مشابه now، اما بدون تنظیم منطقه زمانی). | datetime.today() |
| یک رشته تاریخ/زمان را با استفاده از یک قالب مشخص تبدیل به یک شیء datetime می‌کند. | datetime.strptime() |
| شیء datetime را به یک رشته تاریخ/زمان با قالب مشخص تبدیل می‌کند. | datetime.strftime() |
| تاریخ و زمان معادل یک زمان یونیکس (ثانیه از ۱ ژانویه ۱۹۷۰) را برمی‌گرداند. | datetime.fromtimestamp() |
| شیء datetime را به زمان یونیکس (ثانیه از ۱ ژانویه ۱۹۷۰) تبدیل می‌کند. | datetime.timestamp() |
| یک تاریخ از کلاس date و یک زمان از کلاس time را ترکیب کرده و یک شیء datetime ایجاد می‌کند. | datetime.combine() |
| مقادیر مشخصی در یک شیء datetime را تغییر می‌دهد و شیء جدیدی ایجاد می‌کند. | datetime.replace() |
| زمان UTC معادل یک زمان یونیکس را برمی‌گرداند. | datetime.utcnow() |
| شماره روز هفته را بر اساس استاندارد ISO برمی‌گرداند (۱=دوشنبه، ۷=یکشنبه). | datetime.isoweekday() |
| شماره روز هفته را برمی‌گرداند (۰=دوشنبه، ۶=یکشنبه). | datetime.weekday() |
| یک رشته تاریخ/زمان در فرمت ISO 8601 تولید می‌کند. | datetime.isoformat() |
| اطلاعات تاریخ و زمان را به صورت یک شیء time.struct_time (برای سازگاری با ماژول time) برمی‌گرداند. | datetime.timetuple() |
| شیء datetime را به یک منطقه زمانی خاص تبدیل می‌کند. | datetime.astimezone() |
| قسمت تاریخ از شیء datetime را به عنوان یک شیء date برمی‌گرداند. | datetime.date() |
| قسمت زمان از شیء datetime را به عنوان یک شیء time برمی‌گرداند. | datetime.time() |
| افست منطقه زمانی UTC مربوط به یک شیء datetime را برمی‌گرداند. | datetime.utcoffset() |

| | |
|---|----------------------|
| تغییرات ساعت تابستانی (Daylight Saving Time) را برمی‌گرداند (در صورت وجود). | datetime.dst() |
| سال را به صورت یک عدد صحیح برمی‌گرداند. | datetime.year |
| ماه را به صورت یک عدد صحیح (۱ تا ۱۲) برمی‌گرداند. | datetime.month |
| روز ماه را به صورت یک عدد صحیح برمی‌گرداند. | datetime.day |
| ساعت را به صورت یک عدد صحیح (۰ تا ۲۳) برمی‌گرداند. | datetime.hour |
| دقیقه را به صورت یک عدد صحیح (۰ تا ۵۹) برمی‌گرداند. | datetime.minute |
| ثانیه را به صورت یک عدد صحیح (۰ تا ۵۹) برمی‌گرداند. | datetime.second |
| میکروثانیه را به صورت یک عدد صحیح (۰ تا ۹۹۹۹۹۹) برمی‌گرداند. | datetime.microsecond |

۱۵-۴) توابع مهم و پرکاربرد مازول Datetime

۱۵-۴-۱) برگرداندن تاریخ و زمان فعلی سیستم

```
from datetime import datetime
now = datetime.now()
print(now)
```

خروجی:

۱۵-۴-۲) برگرداندن تاریخ و زمان فعلی سیستم بدون تنظیمات منطقه زمانی

```
from datetime import datetime
```

```
today = datetime.today()  
print(today)
```

خروجی:

۱۵-۴-۳) برگرداندن تاریخ و زمان فعلی در قالب UTC

```
from datetime import datetime  
  
utc_now = datetime.utcnow()  
  
print(utc_now)
```

خروجی:

۱۵-۴-۴) تبدیل یک رشته تاریخ/زمان به یک شیء datetime

```
from datetime import datetime  
  
date_str = "2024-12-29 15:45"  
  
date_obj = datetime.strptime(date_str, "%Y-%m-%d %H:%M")  
  
print(date_obj)
```

خروجی:

۱۵-۴-۵) تبدیل یک شیء datetime را به رشته تاریخ/زمان

```
from datetime import datetime
```

```
now = datetime.now()  
formatted = now.strftime("%Y/%m/%d %H:%M:%S")  
print(formatted)
```

خروجی:

(۱۹۷۰ زانویه ۱ از ثانیه‌ها تعداد زمان یونیکس معادل زمان و تاریخ برگرداندن ۴-۶-۱۵)

```
from datetime import datetime  
  
timestamp = 1709210700  
  
date_from_timestamp = datetime.fromtimestamp(timestamp)  
  
print(date_from_timestamp)
```

خروجی:

(۱۵-۴-۲۰۲۴) شیء **datetime** یک زمان و تاریخ ترکیب ایجاد یک

```
from datetime import datetime, date, time  
  
d = date(2024, 12, 29)  
  
t = time(15, 45)  
  
combined = datetime.combine(d, t)  
  
print(combined)
```

خروجی:

۸-۴-۱۵) تغییر دادن قسمتی از یک شیء **datetime**

```
from datetime import datetime  
  
now = datetime.now()  
  
new_time = now.replace(hour=20, minute=30)  
  
print(new_time)
```

خروجی:

۹-۴-۱۵) برگرداندن زمان یونیکس معادل یک شیء **datetime**

```
from datetime import datetime  
  
now = datetime.now()  
  
timestamp = now.timestamp()  
  
print(timestamp)
```

خروجی:

۱۰-۴-۱۵) تبدیل یک شیء **datetime** به رشته‌ای در قالب ISO 8601

```
from datetime import datetime  
  
now = datetime.now()
```

```
iso_format = now.isoformat()
```

```
print(iso_format)
```

خروجی:

فصل شانزدهم

آشنایی با مژول نامپای در پایتون

۱۶) آشنایی با ماژول نامپای(NumPy) در پایتون

ماژول نامپای 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 (ndarray) و مجموعه‌ای از توابع ریاضی، آماری و جبری، یک ابزار کلیدی برای دانشمندان داده، مهندسان، و تحلیل‌گران است.

۱۶-۱) ویژگی‌های اصلی نامپای

۱. آرایه‌های چندبعدی: هسته نامپای بر پایه آرایه‌های چندبعدی بنا شده است. آرایه‌های نامپای از لیست‌های

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.(vectorized operations).

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.

۴. عملیات ماتریسی و جبر خطی: نامپای ابزارهای گسترده‌ای 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 (linspace) 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 لیست‌های پایتون هستند.
- نامپای از ابزارهایی برای تغییر شکل، برش و انجام محاسبات ریاضی برخوردار است که به‌سادگی با لیست‌های 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 پروژه‌های 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 اجرا کنید:

`pip install numpy`

۲-۵-۱۶) بررسی نصب مژول نامپای

پس از نصب، برای اطمینان از نصب صحیح، دستور زیر را 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 مژول نامپای

برای نصب یک نسخه مشخص از نامپای:

```
pip install numpy==1.22.0
```

۱۶-۵-۴) بروزرسانی ماثول نامپای

برای بروزرسانی به آخرین نسخه:

```
pip install --upgrade numpy
```

۱۶-۵-۵) نصب ماثول نامپای در محیط مجازی

اگر در یک محیط مجازی (Virtual Environment) کار می‌کنید، ابتدا محیط مجازی را فعال کرده و سپس نامپای را نصب کنید:

```
pip install numpy
```

۱۶-۶) 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.

۱۶-۶-۱) ویژگی‌های ماتریس‌ها در نامپای

۱. ساختار دوبعدی: ماتریس‌ها همیشه دارای دو بعد هستند (سطر و ستون).
۲. عملیات بوداری و اسکالار: 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 دو بعدی 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 مازول نامپای
برای ایجاد ماتریس‌ها، از آرایه‌های دو بعدی استفاده می‌کنیم.
روش‌های ایجاد ماتریس:

۱-۶-۲) از لیست‌های تو در تو
import numpy as np
mat = np.array([[1, 2], [3, 4]])

۲-۶-۲) ایجاد ماتریس‌های خاص:
ماتریس صفر:

np.zeros((2, 3))

ماتریس یک:

np.ones((3, 3))

ماتریس واحد (قطری):

ماتریس np.eye(3) # ۳*۳

۱۶-۳-۶) عملیات‌های اصلی ماتریسی

۱۶-۳-۶-۱) جمع و تفریق ماتریس‌ها

انجام عملیات جمع و تفریق به صورت عنصر به عنصر:

$A + B$

$A - B$

۱۶-۳-۶-۲) ضرب ماتریسی

برای ضرب ماتریسی، از تابع وابسته `np.dot()` یا عملگر `@` استفاده می‌شود:

`np.dot(A, B)`

$A @ B$

ضرب عنصر به عنصر

$A * B$

۱۶-۳-۶-۳) ترانهاده (Transpose)

تغییر سطرها به ستون‌ها:

$A.T$

۱۶-۴-۶) معکوس ماتریس

محاسبه معکوس با تابع `:np.linalg.inv()`

`np.linalg.inv(A)`

۱۶-۵-۶) دترمینان ماتریس

محاسبه دترمینان با `:np.linalg.det()`

```
np.linalg.det(A)
```

۶-۶-۱۶) محاسبه مقادیر ویژه و بردارهای ویژه

با تابع وابسته `:np.linalg.eig()`

```
eigenvalues, eigenvectors = np.linalg.eig(A)
```

۷-۶-۱۶) دسترسی به عناصر ماتریس

۱-۷-۶-۱۶) دسترسی به عنصر خاص

عنصر سطر اول و ستون دوم #

۲-۷-۶-۱۶) دسترسی به یک سطر یا ستون

A[0, :] # سطر اول

A[:, 1] # ستون دوم

۸-۶-۱۶) تغییر شکل ماتریس:

۱-۸-۶-۱۶) تغییر شکل ماتریس با `reshape`

A.reshape((3, 2)) تغییر شکل ماتریس به ۳ سطر و ۲ ستون #

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:

```
import numpy as np

# آرایه یکبعدی
array_1d = np.array([1, 2, 3, 4])
print(array_1d) # [1 2 3 4]

# آرایه دو بعدی
array_2d = np.array([[1, 2], [3, 4]])
print(array_2d)
```

خروجی:

۲-۷-۱۶) ایجاد آرایه‌های خاص

مثال:

```
import numpy as np

# آرایه‌ای از صفرها
zeros = np.zeros((2, 3))
```

```
print(zeros)  
# آرایه‌ای از یک‌ها  
  
ones = np.ones((3, 3))  
  
print(ones)  
  
# آرایه‌ای از اعداد متوالی  
  
sequence = np.arange(1, 10, 2)  
  
print(sequence) # [1 3 5 7 9]  
  
# آرایه‌ای از اعداد بین دو مقدار  
  
linspace = np.linspace(0, 1, 5)  
  
print(linspace)
```

خروجی:

۱۶-۷-۳) شکل و اندازه آرایه

مثال:

```
import numpy as np  
  
array = np.array([[1, 2, 3], [4, 5, 6]])  
  
print(array.shape) # (2, 3)
```

```
print(array.size) # 6  
print(array.ndim) # 2 (بعد آرایه)
```

خروجی:

۱۶-۷-۴) عملیات عددی

مثال:

```
import numpy as np  
  
a = np.array([1, 2, 3])  
  
b = np.array([4, 5, 6])
```

جمع و تفریق

```
print(a + b) # [5 7 9]  
print(a - b) # [-3 -3 -3]
```

ضرب و تقسیم

```
print(a * b) # [ 4 10 18]  
print(a / b) # [0.25 0.4 0.5 ]
```

توان

```
print(a ** 2) # [1 4 9]
```

خروجی:

١٦-٧-٥) توابع ریاضی

مثال:

```
import numpy as np
```

```
array = np.array([1, 2, 3, 4])
```

```
print(np.sum(array)) # 10
```

```
print(np.mean(array)) # 2.5
```

```
print(np.max(array)) # 4
```

```
print(np.min(array)) # 1
```

```
print(np.std(array)) # 1.118033988749895
```

خروجی:

۶-۷-۱۶) ایندکس و برش

مثال:

```
import numpy as np  
  
array = np.array([[1, 2, 3], [4, 5, 6]])  
  
# دسترسی به عنصر خاص  
print(array[0, 1]) # 2  
  
# برش آرایه  
print(array[:, 1]) # [2 5]  
  
print(array[0, :]) # [1 2 3]
```

خروجی:

۷-۷-۱۶) تغییر شکل آرایه

مثال:

```
import numpy as np  
  
array = np.array([1, 2, 3, 4, 5, 6])  
  
reshaped = array.reshape((2, 3))  
  
print(reshaped)
```

```
# [[1 2 3]
```

```
# [4 5 6]]
```

خروجی:

۱۶-۷-۸) عمليات ماتريسي

مثال:

```
import numpy as np
```

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

```
b = np.array([[5, 6], [7, 8]])
```

ضرب ماتريسي

```
print(np.dot(a, b))
```

```
# [[19 22]
```

```
# [43 50]]
```

ترانهاده

```
print(a.T)
```

```
# [[1 3]
```

```
# [2 4]]
```

خروجی:

فصل هفدهم

آشنایی با ماژول SymPy در پایتون

۱۷ آشنایی با ماژول SymPy در پایتون

ماژول **SymPy** یک کتابخانه‌ی نمادین (**symbolic**) در پایتون است که برای انجام محاسبات ریاضی نمادین برنامه‌نویسی پایتون است که برای انجام محاسبات ریاضی نمادین (**symbolic computation**) طراحی شده است. **SymPy** یک کتابخانه متن‌باز (**open-source**) در زبان (**Symbolic Mathematics**) برنامه‌نویسی پایتون است که برای انجام محاسبات ریاضی نمادین (**CAS: Computer Algebra System**) عمل می‌کند و به عنوان یک سیستم جبری کامپیوتری (Computer Algebra System) می‌باشد. این کتابخانه به عنوان یک سیستم جبری کامپیوتری (Computer Algebra System) 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 تمرکز دارد. برخلاف کتابخانه‌هایی مانند نامپای که محاسبات عددی انجام می‌دهند، **SymPy** می‌تواند عبارات ریاضی را به صورت نمادین مانند $y = x + 1$ 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 پروژه 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 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 انتگرال گیری 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 مشتقهای 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.

Sympy (۳-۱۷) نقاط قوت

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 یا Mathematica رایگان و 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 از دستور استفاده کنید:

import SymPy

۵-۱۷) امکانات SymPy

۱-۵-۱۷) ایجاد متغیرهای نمادین

با استفاده از SymPy ، متغیرهای ریاضی را به صورت نمادین تعریف می‌کنید:

```
from sympy import symbols
```

```
x, y = symbols('x y')
```

۲-۵-۱۷) ساده‌سازی و گسترش عبارات

ابزارهایی برای ساده‌سازی و بازنویسی عبارات ریاضی:

۱-۲-۵-۱۷) SymPy ساده‌سازی عبارات در مazzoل

```
from sympy import simplify  
  
expr = (x**2 + 2*x + 1)/(x + 1)  
  
simplify(expr) # x + 1
```

۲-۲-۵-۱۷) گسترش عبارات در مazzoل SymPy

```
from sympy import expand  
  
expand((x + 1)**2) # x**2 + 2*x + 1
```

۳-۵-۱۷) مشتق و انتگرال

۱-۳-۵-۱۷) محاسبه مشتق

```
from sympy import diff  
  
diff(x**2 + 3*x + 5, x) # 2*x + 3
```

۲-۳-۵-۱۷) محاسبه انتگرال

```
from sympy import integrate  
  
integrate(x**2, x) # x**3 / 3
```

۴-۵-۱۷) حل معادلات

۱-۴-۵-۱۷) حل معادلات جبری

```
from sympy import solve  
  
solve(x**2 - 4, x) # [2, -2]
```

۲-۴-۵-۱۷) حل معادلات دیفرانسیل

```
from sympy import Function, Eq, dsolve
```

```

f = Function('f')

eq = Eq(f(x).diff(x, 2) - 3*f(x), 0)

dsolve(eq) # حل معادله دیفرانسیل

```

۱۷-۵-۵) رسم نمودار

قابلیت رسم نمودار عبارات را نیز دارد:

```

from sympy.plotting import plot

plot(x**2)

```

۱۷-۶) تفاوت NumPy با SymPy

- SymPy برای محاسبات نمادین طراحی شده است و می‌تواند عبارات ریاضی را به صورت سمبلیک مدیریت کند.
- NumPy برای محاسبات عددی طراحی شده و برای داده‌های عددی بزرگ مناسب است.
- SymPy یک ابزار قدرتمند برای کسانی است که با ریاضیات پیشرفت‌های سروکار دارند و نیاز به تحلیل و حل مسائل به صورت نمادین دارند.

۱۷-۷) مثال‌های کاربردی SymPy

در اینجا چندین مثال کاربردی از SymPy برای نشان دادن توانایی‌های آن در مسائل ریاضی آورده شده است:

۱۷-۸) ساده‌سازی عبارات ریاضی

مثال:

```
from sympy import symbols, simplify
```

```
x = symbols('x')  
  
expr = (x**2 + 2*x + 1) / (x + 1)  
  
result = simplify(expr)  
  
print(result) # خروجی: x + 1
```

خروجی:

۱۷-۷-۲) حل معادلات جبری (حل معادله درجه دوم)

مثال:

```
from sympy import symbols, solve  
  
x = symbols('x')  
  
eq = x**2 - 4  
  
roots = solve(eq, x)  
  
print(roots) # خروجی: [2, -2]
```

خروجی:

۱۷-۷-۳) محاسبه مشتق یک تابع

مثال:

```
from sympy import symbols, diff  
  
x = symbols('x')  
  
expr = x**3 + 2*x**2 + x  
  
derivative = diff(expr, x)  
  
print(derivative) # خروجی: ۳*x**2 + 4*x + 1
```

خروجی:

۱۷-۷-۴) انتگرال‌گیری از یک تابع

مثال:

```
from sympy import symbols, integrate  
  
x = symbols('x')  
  
expr = x**2  
  
integral = integrate(expr, x)  
  
print(integral) # خروجی: x**3 / 3
```

خروجی:

۱۷-۷-۵) حل معادلات دیفرانسیل

مثال:

```
from sympy import Function, Eq, dsolve, symbols  
x = symbols('x')  
f = Function('f')  
eq = Eq(f(x).diff(x, 2) - 3*f(x), 0)  
solution = dsolve(eq)  
print(solution)
```

خروجی:

۱۷-۷-۶) کار با ماتریس‌ها (محاسبه دترمینان یک ماتریس)

مثال:

```
from sympy import Matrix  
A = Matrix([[1, 2], [3, 4]])  
determinant = A.det()  
print(determinant) # ۲-
```

خروجی:

خروجی:

۷-۷-۱۷) محاسبه مقادیر ویژه و بردارهای ویژه

مثال:

```
from sympy import Matrix  
  
A = Matrix([[2, 0], [0, 3]])  
  
eigenvalues = A.eigenvals()  
  
eigenvectors = A.eigenvects()  
  
print(eigenvalues) # خروجی: {2: 1, 3: 1}
```

خروجی: اطلاعات بردارهای ویژه

خروجی:

۸-۷-۱۷) سری تیلور

مثال:

```
from sympy import symbols, sin, series  
  
x = symbols('x')  
  
expr = sin(x)
```

```
taylor = series(expr, x, 0, 6)  
print(taylor) # خروجی:  $x - x^{*3}/6 + x^{*5}/120 + O(x^{*6})$ 
```

خروجی:

۱۷-۷-۹) رسم نمودار یک تابع

مثال:

```
from sympy.plotting import plot  
from sympy import symbols  
x = symbols('x')  
expr = x**2 - 4  
plot(expr) # رسم نمودار تابع
```

خروجی:

۱۷-۷-۱۰) کاربرد در هندسه (محاسبه مساحت دایره)

مثال:

```
from sympy import symbols, pi  
r = symbols('r')  
area = pi * r**2  
print(area) # خروجی:  $\pi * r^{*2}$ 
```

خروجی:

فصل هجدهم

آشنایی با ماثول pandas در پایتون

۱۸) آشنایی با ماژول pandas در پایتون

یک کتابخانه مترباز 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 (Data Science)، 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) ویژگی‌های اصلی Pandas

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 (missing data).

۳. ابزارهای تحلیل داده

گروه‌بندی (grouping) 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 (Time-Series Analysis).
- قابلیت تولید گزارش‌های خلاصه از داده‌ها.

۴. انعطاف‌پذیری بالا

- کار با مجموعه‌های داده کوچک و بزرگ.
- عملکرد عالی در تحلیل داده‌های زمانی (time-series analysis).
- 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
- قابلیت تبدیل داده‌ها به ساختارهای دیگر برای استفاده در ابزارهای تخصصی.

۶. کارایی بالا

- این کتابخانه از هسته NumPy بهره می‌برد، که باعث افزایش کارایی در پردازش داده‌ها می‌شود.
- امکان مدیریت داده‌های حجمی بهینه شده برای حافظه.

۷. رابط کاربری ساده

به گونه‌ای 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 (Data Cleaning) : حذف داده‌های گمشده، تصحیح مقادیر نادرست و تبدیل داده‌ها

به قالب مناسب.

۲. تحلیل داده‌ها (Data Analysis) : انجام تحلیل‌های آماری و جمع‌آوری اطلاعات مفید از داده‌ها.

۳. کاوش داده‌ها (Data Exploration) : مشاهده خلاصه‌ای از داده‌ها، توزیع مقادیر، و ویژگی‌های کلیدی.

۴. مدیریت داده‌های زمانی (Time-Series Data) : 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.

۶. تبدیل داده‌ها: تبدیل داده‌ها به فرمتهای قابل استفاده برای مدل‌های یادگیری ماشین.

۳-۱۸) مزایای استفاده از Pandas

• سادگی: 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 توسعه‌دهندگان فعال دارد.

- پشتیبانی از داده‌های حجمی: Pandas می‌تواند داده‌های حجمی را که در حافظه قرار می‌گیرند، با کارایی بالا مدیریت کند.

۴-۱۸ مقایسه Pandas با سایر کتابخانه‌ها

| Excel | NumPy | Pandas | ویژگی |
|------------------------|-------------------------|---------------------------------|-------------|
| صفحات گسترده | آرایه‌های چندبعدی | ساختارهای داده‌ای پیشرفته | نوع داده‌ها |
| مدیریت داده‌های ساده | محاسبات عددی | تحلیل و مدیریت داده‌ها | کاربرد اصلی |
| محدود به داده‌های کوچک | سریع برای داده‌های عددی | بهینه برای داده‌های ساختاریافته | عملکرد |

۵-۱۸ نصب و وارد کردن ماثول Pandas به پروژه

برای نصب ماثول Pandas از دستور زیر استفاده کنید:

pip install pandas

برای وارد کردن ماثول pandas به پروژه از دستور زیر استفاده می‌شود:

import pandas

۶-۱۸) مثال‌های کاربردی مازول Pandas در پایتون

در اینجا چند مثال کاربردی برای استفاده از کتابخانه Pandas آورده شده است که کاربردهای آن در تحلیل داده را نشان می‌دهند:

۱-۶-۱۸) بارگذاری داده‌ها از فایل CSV و نمایش داده‌ها

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
print(df.head()) # نمایش اولین چند ردیف
```

خروجی:

۲-۶-۱۸) تمیز کردن داده‌ها (شناسایی و حذف داده‌های گمشده)

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
df.dropna(inplace=True) # حذف ردیف‌هایی که مقادیر گمشده دارند
```

خروجی:

۳-۶-۱۸) مرتب‌سازی داده‌ها (مرتب‌سازی داده‌ها بر اساس یک ستون)

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
df.sort_values('column_name', ascending=False, inplace=True)
```

خروجی:

۴-۶-۱۸) گروه‌بندی داده‌ها (محاسبه مجموع داده‌ها در گروه‌های مختلف)

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
grouped = df.groupby('category_column').sum()  
  
print(grouped)
```

خروجی:

۵-۶-۱۸) محاسبات آماری (محاسبه میانگین، کمینه و بیشینه داده‌ها)

مثال:

```
import pandas as pd
```

```
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
mean_value = df['numeric_column'].mean()  
  
min_value = df['numeric_column'].min()  
  
max_value = df['numeric_column'].max()
```

خروجی:

۱۸-۶) انتخاب داده‌ها (فیلتر کردن داده‌ها بر اساس یک شرط)

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
filtered_data = df[df['column_name'] > 50]
```

خروجی:

۱۸-۷) ایجاد ستون جدید (ایجاد یک ستون جدید بر اساس محاسبات ستون‌های موجود)

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
df['new_column'] = df['column1'] + df['column2']
```

خروجی:

۱۸-۶-۸) ترکیب و ادغام داده‌ها

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
merged_df = pd.merge(df1, df2, on='common_column')
```

خروجی:

۱۸-۶-۹) تغییر فرمت داده‌ها (تبدیل تاریخ‌ها به فرمت زمانی)

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
df['date_column'] = pd.to_datetime(df['date_column'])
```

خروجی:

۱۸-۶-۱۰) تحلیل داده‌های زمانی (محاسبه داده‌ها بر اساس بازه‌های زمانی)

مثال:

```
import pandas as pd
```

```
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
df.set_index('date_column', inplace=True)  
  
monthly_data = df.resample('M').sum()
```

خروجی:

(۱۸-۶-۱۱) ذخیره داده‌ها (ذخیره داده‌های ویرایش شده به یک فایل جدید)

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
df.to_csv('cleaned_data.csv', index=False)
```

خروجی:

(۱۸-۶-۱۲) مصورسازی ساده با **Pandas** (رسم نمودار از داده‌ها)

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
df['column_name'].plot(kind='line')
```

خروجی:

۱۳-۶-۱۸) محاسبه درصدها (محاسبه نسبت درصدی یک ستون)

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
df['percentage'] = (df['column_name'] / df['column_name'].sum()) * 100
```

خروجی:

۱۴-۶-۱۸) حذف ستون‌ها

مثال:

```
import pandas as pd  
  
df = pd.read_csv('data.csv') # بارگذاری فایل  
  
df.drop('column_name', axis=1, inplace=True)
```

خروجی:

۱۵-۶-۱۸) شناسایی و حذف داده‌های تکراری

مثال:

```
import pandas as pd
```

```
df = pd.read_csv('data.csv') # بارگذاری فایل  
df.drop_duplicates(inplace=True)
```

خروجی:

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.

فصل نوزدهم

آشنایی با مازول **matplotlib** در پایتون

(۱۹) آشنایی با مژول **Matplotlib** در پایتون

Matplotlib یکی از کتابخانه‌های قدرتمند و محبوب در پایتون است که برای مصورسازی داده‌ها و ایجاد نمودارهای متنوع 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 به طور 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 رابطه‌های 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 به طور 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.

Matplotlib زیرماژول‌های (۲-۱۹)

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.

Matplotlib کاربردهای (۳-۱۹)

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 (Line Plot)

برای رسم نمودار خطی که معمولاً برای نمایش تغییرات یک متغیر استفاده می‌شود:

```
import matplotlib.pyplot as plt
```

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

```
y = [2, 4, 6, 8, 10]
```

```
plt.plot(x, y, label='Linear Relationship', color='blue', marker='o')
```

```
عنوان نمودار # plt.title('Line Plot Example')
```

```
plt.xlabel('X-axis') # برچسب محور X
```

```
plt.ylabel('Y-axis') # برچسب محور Y
```

```
plt.legend() # اضافه کردن راهنمایی
```

```
plt.grid(True) # نمایش خطوط شبکه
```

```
plt.show()
```

خروجی:

(Bar Plot) ترسیم نمودار میله‌ای (۱۹-۴-۲)

برای نمایش مقایسه مقادیر دسته‌بندی شده:

```
import matplotlib.pyplot as plt
```

```
categories = ['A', 'B', 'C', 'D']
```

```
values = [10, 15, 7, 12]
```

```
plt.bar(categories, values, color='green')
```

```
plt.title('Bar Plot Example')
```

```
plt.xlabel('Categories')
```

```
plt.ylabel('Values')
```

```
plt.show()
```

خروجی:

(Pie Chart) ترسیم نمودار دایره‌ای (۳-۴-۱۹)

برای نمایش نسبت‌ها و درصدها:

```
import matplotlib.pyplot as plt

labels = ['Category A', 'Category B', 'Category C', 'Category D']

sizes = [25, 35, 20, 20]

explode = (0, 0.1, 0, 0) # برجسته کردن بخش دوم

plt.pie(sizes, labels=labels, autopct='%.1f%%', explode=explode, shadow=True,
startangle=90)

plt.title('Pie Chart Example')

plt.show()
```

خروجی:

(Scatter Plot) ترسیم نمودار پراکندگی (۴-۴-۱۹)

برای نمایش رابطه بین دو متغیر:

```
import numpy as np

import matplotlib.pyplot as plt
```

```
x = np.random.rand(50)

y = np.random.rand(50)

plt.scatter(x, y, color='red', alpha=0.7)

plt.title('Scatter Plot Example')

plt.xlabel('X-axis')

plt.ylabel('Y-axis')

plt.show()
```

خروجی:

۱۹-۴-۵) ترسیم هیستوگرام (Histogram)

برای نمایش توزیع داده‌ها:

```
import matplotlib.pyplot as plt

data = [1, 2, 2, 3, 3, 3, 4, 4, 4, 4, 5, 5]

plt.hist(data, bins=5, color='purple', edgecolor='black')

plt.title('Histogram Example')
```

```
plt.xlabel('Value')  
plt.ylabel('Frequency')  
plt.show()
```

خروجی:

۶-۴-۱۹) ترسیم چند نمودار در یک صفحه

برای نمایش چند نمودار به صورت جداگانه در یک صفحه:

```
import matplotlib.pyplot as plt
```

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

```
y1 = [1, 4, 9, 16, 25]
```

```
y2 = [2, 4, 6, 8, 10]
```

یک ردیف، دو ستون، نمودار اول #

```
plt.plot(x, y1, color='blue')
```

```
plt.title('Subplot 1')
```

یک ردیف، دو ستون، نمودار دوم #

```
plt.plot(x, y2, color='green')

plt.title('Subplot 2')

plt.tight_layout() # تنظیم فاصله‌ها
```

plt.show()

خروجی:

(۳D Plot) ترسیم نمودار سه‌بعدی (۱۹-۴-۷)

برای نمایش داده‌های سه‌بعدی:

```
import matplotlib.pyplot as plt

from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()

ax = fig.add_subplot(111, projection='3d')

x = [1, 2, 3, 4, 5]

y = [2, 4, 6, 8, 10]

z = [1, 3, 5, 7, 9]
```

```
ax.plot(x, y, z, label='3D Line')

ax.set_title('3D Plot Example')

ax.set_xlabel('X-axis')

ax.set_ylabel('Y-axis')

ax.set_zlabel('Z-axis')

plt.show()
```

خروجی:

۸-۴-۱۹) تنظیمات پیشفرض

- رنگها و خطوط: می‌توانید از انواع رنگها ('red', '#FF5733') و خطوط (':', '--') استفاده کنید.
- ذخیره نمودار: برای ذخیره نمودار به صورت فایل:

```
plt.savefig('plot.png', dpi=300)
```

۵-۱۹) محورها و برچسبها در ماژول Matplotlib

در **Matplotlib**, **محورها (Axes)** و **برچسبها (Labels)** 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 (**X-axis**) و عمودی (**Y-axis**) از توابع زیر استفاده می‌شود:

• plt.xlabel(label, **kwargs): برای برچسب‌گذاری محور افقی.

• plt.ylabel(label, **kwargs): برای برچسب‌گذاری محور عمودی.

مثال:

```
import matplotlib.pyplot as plt
```

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

```
y = [2, 4, 6, 8, 10]
```

```
plt.plot(x, y)
```

```
plt.xlabel('Time (s)', fontsize=12, color='blue') # برچسب محور افقی
```

```
plt.ylabel('Distance (m)', fontsize=12, color='green') # برچسب محور عمودی
```

```
plt.title('Motion Analysis') # عنوان نمودار
```

```
plt.show()
```

خروجی:

۲-۵-۱۹) تنظیم محدوده محورها

• plt.xlim([min, max]): تنظیم محدوده محور افقی.

• plt.ylim([min, max]): تنظیم محدوده محور عمودی.

مثال:

```
plt.plot(x, y)
```

```
plt.xlim(0, 6) # محدوده محور X
```

```
plt.ylim(0, 12) # محدوده محور Y
```

```
plt.show()
```

خروجی:

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:

```
plt.plot(x, y)
```

```
plt.grid(True, linestyle='--', color='gray', alpha=0.7)
```

```
plt.show()
```

خروجی:

۱۹-۵-۴) 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:

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

```
labels_x = ['One', 'Two', 'Three', 'Four', 'Five']
```

```
ticks_y = [2, 4, 6, 8, 10]
```

```
labels_y = ['Low', 'Medium', 'High', 'Higher', 'Highest']
```

```
plt.plot(x, y)
```

```
X تغییر مقادیر محور plt.xticks(ticks_x, labels_x) #
```

```
Y تغییر مقادیر محور plt.yticks(ticks_y, labels_y) #
```

```
plt.show()
```

خروجی:

۵-۵-۱۹) چرخش برچسبهای محورها

برای چرخاندن برچسبهای محور، از آرگومان `rotation` استفاده می‌شود:

```
plt.plot(x, y)
```

```
X چرخش برچسبهای محور # plt.xticks(rotation=45)
```

```
Y چرخش برچسبهای محور # plt.yticks(rotation=90)
```

```
plt.show()
```

خروجی:

۶-۵-۱۹) افزودن خطوط به محورها

- `plt.axhline()`: رسم یک خط افقی روی نمودار.

- `plt.axvline()`: رسم یک خط عمودی روی نمودار.

- `plt.axhspan()`: افزودن یک ناحیه افقی.

- `plt.axvspan()`: افزودن یک ناحیه عمودی.

مثال:

```
plt.plot(x, y)
```

```
plt.axhline(y=6, color='red', linestyle='--') # خط افقی  
plt.axvline(x=3, color='blue', linestyle='--') # خط عمودی  
plt.show()
```

خروجی:

۷-۵-۱۹) اضافه کردن عناوین به محورها

برای اضافه کردن توضیحاتی دقیق‌تر به محورها می‌توان از عناوین ثانویه استفاده کرد:

- ax.set_xlabel(): افزودن برچسب به محور افقی (در سطح پیشرفته‌تر با استفاده از شیء محور).

- ax.set_ylabel(): افزودن برچسب به محور عمودی.

مثال:

```
fig, ax = plt.subplots()  
  
ax.plot(x, y)  
  
ax.set_xlabel('Time (s)')  
  
ax.set_ylabel('Distance (m)')  
  
plt.show()
```

خروجی:

۸-۵-۱۹) تنظیمات گرافیکی پیشرفته محورها

- تغییر اندازه فونت‌ها با `fontsize`
- تغییر رنگ متن‌ها با `color`
- استفاده از متن‌های ریاضی با استفاده از `\LaTeX`:

```
plt.xlabel(r'$Time\ (s)$', fontsize=14)
```

```
plt.ylabel(r'$Distance\ (m)$', fontsize=14)
```

۶-۱۹) ایجاد چندین نمودار در یک صفحه

در `Matplotlib`, از تابع `subplot()` برای ایجاد چندین نمودار (`plot`) 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 (`grid`) قرار می‌گیرد و می‌توان آن‌ها را به صورت مرتب و سازماندهی شده نمایش داد.

۱-۶-۱۹) `subplot()` ساختار کلی

تابع `subplot()` آرگومان می‌گیرد که تعیین می‌کنند نمودارها چگونه در یک صفحه سازماندهی شوند:

```
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 رسم دو نمودار در یک ردیف و دو ستون:

```
import matplotlib.pyplot as plt
```

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

```
y1 = [1, 4, 9, 16, 25]
```

```
y2 = [2, 4, 6, 8, 10]
```

یک ردیف، دو ستون، نمودار اول #

```
plt.plot(x, y1, color='blue')
```

```
plt.title('Plot 1')
```

یک ردیف، دو ستون، نمودار دوم #

```
plt.plot(x, y2, color='green')
```

```
plt.title('Plot 2')
```

تنظیم فاصله‌ها بین نمودارها #

```
plt.tight_layout()
```

خروجی:

۱۹-۶-۳) چند نمودار در چند ردیف و ستون

مثال رسم چهار نمودار در دو ردیف و دو ستون:

دو ردیف، دو ستون، نمودار اول #

```
plt.plot(x, y1, color='red')
```

```
plt.title('Plot 1')
```

دو ردیف، دو ستون، نمودار دوم #

```
plt.plot(x, y2, color='blue')
```

```
plt.title('Plot 2')
```

دو ردیف، دو ستون، نمودار سوم #

```
plt.bar(x, y1, color='purple')
```

```
plt.title('Plot 3')
```

دو ردیف، دو ستون، نمودار چهارم #

```
plt.scatter(x, y2, color='green')
```

```
plt.title('Plot 4')
```

```
plt.tight_layout() # تنظیم خودکار فاصله‌ها
```

```
plt.show()
```

خروجی:

۱۹-۶) استفاده از Subplot و Axes برای Figure

در Matplotlib، می‌توان از روش پیشرفته‌تر plt.subplots() برای ایجاد چند نمودار استفاده کرد. این روش کنترل بیشتری روی نمودارها فراهم می‌کند.

```
fig, axes = plt.subplots(2, 2, figsize=(8, 6)) # دو ردیف، دو ستون
```

```
axes[0, 0].plot(x, y1, color='red')
```

```
axes[0, 0].set_title('Plot 1')
```

```
axes[0, 1].bar(x, y2, color='blue')
```

```
axes[0, 1].set_title('Plot 2')
```

```
axes[1, 0].scatter(x, y1, color='green')
```

```
axes[1, 0].set_title('Plot 3')
```

```
axes[1, 1].plot(x, y2, linestyle='--', color='purple')

axes[1, 1].set_title('Plot 4')

plt.tight_layout()

plt.show()
```

خروجی:

۱۹-۶-۵) ترکیب نمودارها با ابعاد مختلف

می‌توان برخی از نمودارها را ادغام کرد و فضای بیشتری به آن‌ها اختصاص داد:

```
plt.subplot(2, 2, (1, 2)) # نمودار اول به اندازه دو ستون
```

```
plt.plot(x, y1, color='orange')
```

```
plt.title('Wide Plot')
```

```
plt.subplot(2, 2, 3)
```

```
plt.bar(x, y2, color='green')
```

```
plt.title('Plot 2')
```

```
plt.subplot(2, 2, 4)
```

```
plt.scatter(x, y1, color='blue')
```

```
plt.title('Plot 3')
```

```
plt.tight_layout()
```

```
plt.show()
```

خروجی:

۱۹-۶) اشتراک‌گذاری محورهای مشترک

در برخی موارد نیاز است محورها بین نمودارها به اشتراک گذاشته شوند:

```
fig, axes = plt.subplots(2, 2, sharex=True, sharey=True, figsize=(8, 6)) # اشتراک محورها
```

```
axes[0, 0].plot(x, y1)
```

```
axes[0, 0].set_title('Plot 1')
```

```
axes[0, 1].scatter(x, y2)
```

```
axes[0, 1].set_title('Plot 2')
```

```
axes[1, 0].bar(x, y1)
```

```

axes[1, 0].set_title('Plot 3')

axes[1, 1].plot(x, y2, linestyle='--')
axes[1, 1].set_title('Plot 4')

plt.tight_layout()

plt.show()

```

خروجی:

۱۹-۶-۷) استفاده از تنظیمات پیشرفتی

- تغییر اندازه کلی نمودارها با **figsize**
- تنظیم فاصله‌ها به صورت دستی با `:plt.subplots_adjust()`
- تنظیم فاصله عمودی و افقی با `plt.subplots_adjust(hspace=0.5, wspace=0.5)` #

نکات:

- استفاده از `tight_layout()` باعث جلوگیری از همپوشانی عناصر می‌شود.
- برای نمودارهای پیچیده‌تر، استفاده از `plt.subplots()` توصیه می‌شود.
- توجه به سازمان‌دهی نمودارها در شبکه برای خوانایی بیشتر ضروری است.

۷-۱۹) نمودارهای هیستوگرام و میله‌ای در ماژول Matplotlib

در Matplotlib، هر دو نمودار Bar Chart (Bar) و Histogram (Hist) برای نمایش داده‌ها استفاده می‌شوند، اما کاربرد و نحوه نمایش آن‌ها متفاوت است.

۱-۷-۱۹) نمودار هیستوگرام(Histogram)

هیستوگرام برای نمایش توزیع فراوانی داده‌های عددی به کار می‌رود. داده‌ها به بازه‌های پیوسته (bins) تقسیم شده و تعداد داده‌های موجود در هر بازه نشان داده می‌شود.

۱-۱-۷-۱۹) 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 (True برای نمایش چگالی).

۴-۷-۱۹) مثال برای نمودار هیستوگرام

```
import matplotlib.pyplot as plt
```

```
import numpy as np
```

```
data = np.random.randn(1000) # تولید داده‌های تصادفی
```

```
plt.hist(data, bins=20, color='blue', alpha=0.7, edgecolor='black')
```

```
plt.title('Histogram Example')
```

```
plt.xlabel('Bins')
```

```
plt.ylabel('Frequency')
```

```
plt.show()
```

خروجی:

۲-۷-۱۹) نمودار میله‌ای (Bar Chart)

نمودار میله‌ای برای نمایش داده‌های دسته‌بندی شده (Categorical Data) 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.bart(): برای رسم میله‌ها به صورت افقی

۳-۲-۷-۱۹ آرگومان‌های نمودار میله‌ای

- color: رنگ میله‌ها.

- width: پهنای میله‌ها.

- align: موقعیت میله‌ها روی محور edge center یا center

۴-۲-۷-۱۹) مثال برای نمودار میله‌ای

```
categories = ['A', 'B', 'C', 'D']
```

```
values = [10, 15, 7, 12]
```

```
plt.bar(categories, values, color='orange', alpha=0.8, edgecolor='black')
```

```
plt.title('Bar Chart Example')
```

```
plt.xlabel('Categories')
```

```
plt.ylabel('Values')
```

```
plt.show()
```

خروجی:

۳-۷-۱۹) تفاوت‌های اصلی بین Bar Chart و Histogram

| Bar Chart | Histogram | ویژگی |
|------------------------------|------------------------------|---------------|
| (Categorical) داده‌های گسسته | (Continuous) داده‌های پیوسته | نوع داده |
| (Categories) دسته‌ها | (Bins) بازه‌ها | محور X |
| 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:

```
data = np.random.randn(1000)
```

```
categories = ['A', 'B', 'C', 'D']
```

```
values = [10, 15, 7, 12]
```

```
plt.subplot(1, 2, 1) # رسم هیستوگرام
```

```
plt.hist(data, bins=20, color='blue', alpha=0.7, edgecolor='black')
```

```
plt.title('Histogram')
```

```
plt.subplot(1, 2, 2) # رسم نمودار میله‌ای  
  
plt.bar(categories, values, color='orange', alpha=0.8, edgecolor='black')  
  
plt.title('Bar Chart')  
  
  
  
plt.tight_layout()  
  
plt.show()
```

خروجی:

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.