

الحمد لله
الرحمن
الرحيم

هنر برنامه نویسی با پایتون

مهندس محمد ایزانلو

تقدیم به :

همسرم و فرزندانم آرش و آرمان

سرشناسه	:	ایزائلو، محمد، ۱۳۵۵-
عنوان و نام پدیدآور	:	هنر برنامه نویسی با پایتون / نویسنده محمد ایزائلو.
مشخصات نشر	:	بجنورد: زبان علم، ۱۴۰۲.
مشخصات ظاهری	:	۲۶۷ ص: مصور، جدول، نمودار.
شابک	:	۹۷۸-۶۲۲-۸۱۲۸-۰۱-۶
وضعیت فهرست نویسی	:	فیپا
یادداشت	:	چاپ قبلی: زبان علم، ۱۴۰۲. (۱۸۲ ص.)
موضوع	:	پایتون (زبان برنامه نویسی کامپیوتر)
	:	Python (Computer programming language)
رده بندی کنگره	:	QA76/73
رده بندی دیویی	:	005/133
شماره کتابشناسی ملی	:	۹۳۵۹۱۷۱
اطلاعات رکورد کتابشناسی	:	فیپا

هنر برنامه نویسی با پایتون

نویسنده : مهندس محمد ایزائلو

صفحه آرایی : تیم طراحی و صفحه آرایی انتشارات زبان علم

طراح جلد: هادی صانعی

ناشر : زبان علم (۰۹۰۱۵۰۳۴۳۵۳)

شابک : ۹۷۸-۶۲۲-۸۱۲۸-۰۱-۶

نوبت چاپ : اول (ویراست اول)

تاریخ چاپ: ۱۴۰۲

تیراژ : ۵۰۰

قیمت: ۱۴۵۰۰۰ تومان



خراسان شمالی، بجنورد (zabaneelm.ir)

فهرست مطالب کتاب

دییاجه

۱

فصل نخست

نصب پایتون و آماده سازی محیط برنامه نویسی

۳

روش نصب

۴

نصب سفارشی

۷

محیط برنامه نویسی و اجرا در Python

۹

فصل دوم

آشنایی با مفهوم متغیر و دیگر واژگان دنیای برنامه نویسی

۱۳

محیط توسعه یکپارچه

۱۴

آشنایی با مفهوم متغیر

۱۵

تعریف متغیر در Python

۲۰

قوانین و اصول نام گذاری متغیرها در پایتون

۲۲

فصل سوم

مستند سازی کد و انواع داده

۲۵

توضیحات چند خطی در پایتون

۲۶

۲۷	دستور print در پایتون
۲۷	انواع داده در Python
۳۰	پیدا کردن نوع داده یک متغیر
۳۰	تعیین نوع داده یک متغیر در Python
۳۱	نوع داده عددی
۳۳	نوع داده منطقی
۳۴	رشته ها در پایتون
۳۵	استاندارد آمریکایی برای تبادل اطلاعات
۳۶	استاندارد جهانی Unicode
۳۹	نگاه فنی تر به Utf-8

فصل چهارم

۴۳	عملگرها
۴۴	انواع عملگرها در پایتون
۴۵	عملگر انتساب یا تخصیص
۴۸	عملگرهای مقایسه ای و نوع داده منطقی
۴۹	عملگر تساوی یا برابری
۵۰	عملگر بزرگتر
۵۰	عملگر کوچکتر
۵۱	عملگر نابرابر

۵۱	عملگر بزرگتر مساوی
۵۲	عملگر کوچکتر مساوی
۵۲	متغیرهای نوع منطقی
۵۴	عملگرهای منطقی
۵۵	عملگر منطقی and
۵۸	عملگر منطقی OR
۶۲	عملگر منطقی not
۶۳	عملگرهای انتساب ترکیبی

فصل پنجم

۶۴	تصمیم گیری و هدایت جریان برنامه
۶۵	دستور if
۶۸	بخش else در دستور if
۷۰	بخش elif در دستور if
۷۴	If های تودرتو
۷۶	دستور match

فصل ششم

آشنایی با دستور RANGE ، عملگرهای عضویت و تبدیل نوع

۷۹	
۸۱	عملگرهای عضویت
۸۱	عملگر in
۸۲	عملگر not in
۸۳	دستور input
۸۵	تبدیل نوع
۸۶	دستورات تبدیل نوع صریح در پایتون
۸۹	تبدیل نوع ضمنی

فصل هفتم

حلقه ها

۹۲	
۹۵	حلقه for
۱۰۱	حلقه while
۱۰۳	حلقه های تودرتو
۱۰۷	دستور break
۱۰۸	دستور continue

فصل هشتم

اولویت یا برتری عملگرها ۱۱۰

فصل نهم

نوع داده list ۱۱۸

۱۲۱ روش دستیابی به عناصر یک فهرست

۱۲۲ طول فهرست

۱۲۳ انتخاب محدوده معینی از یک فهرست

۱۲۵ به روز رسانی فهرست

۱۲۷ بروزرسانی یک محدوده از عناصر موجود در فهرست

۱۲۸ شگرد insert()

۱۲۹ شگرد append()

۱۳۰ شگرد extend()

۱۳۱ حذف عناصر یک فهرست

۱۳۳ حذف تمامی عناصر یک فهرست

۱۳۳ پیمایش فهرست ها

۱۳۴ ساده سازی پیمایش فهرست با شیوه فهرست برگزیده

۱۳۶ مرتب کردن فهرست

۱۳۸ نسخه برداری از یک فهرست

پیوند زدن دو یا چندین فهرست به یکدیگر

۱۴۰

فصل دهم

نوع داده tuple

۱۴۲

روش دستیابی به عناصر یک چندتایی

۱۴۵

طول چندتایی

۱۴۶

انتخاب محدوده معینی از یک چندتایی

۱۴۷

بروزرسانی مقدار یک عنصر در tuple

۱۴۹

افزودن یک عنصر به tuple

۱۴۹

حذف یک عنصر از tuple

۱۵۰

بسته بندی و بازپخش چندتایی ها

۱۵۱

پیمایش چندتایی ها

۱۵۳

الحاق یا به هم پیوستن چندتایی ها

۱۵۴

شگرد های پرکاربرد در چندتایی ها

۱۵۵

شگرد count()

۱۵۵

شگرد index()

۱۵۵

تابع zip()

۱۵۵

فصل یازدهم

نوع داده dictionary

۱۵۸

- ۱۶۱ ویژگی های واژه نامه
- ۱۶۱ روش دستیابی به عناصر یک واژه نامه
- ۱۶۳ بدست آوردن کلید های یک dictionary در قالب یک شی قابل پیمایش
- ۱۶۴ بدست آوردن مقدار های یک dictionary در قالب یک شی قابل پیمایش
- ۱۶۴ طول واژه نامه
- ۱۶۶ به روزرسانی مقدار یک عنصر در dictionary
- ۱۶۷ افزودن یک عنصر به dictionary
- ۱۶۸ حذف یک عنصر از dictionary
- ۱۷۰ پیمایش واژه نامه
- ۱۷۲ نسخه برداری از یک واژه نامه
- ۱۷۳ واژه نامه های تو در تو

فصل دوازدهم

- ۱۷۶ تابع
- ۱۷۸ نوشتن تابع در پایتون
- ۱۸۱ فراخوانی یک تابع
- ۱۸۳ آرگومانهای نامحدود
- ۱۸۵ ارسال آرگومانها با استفاده از نام پارامتر
- ۱۸۷ پارامترهای نامدار نامحدود
- ۱۸۹ پارامتر های دارای مقدار پیش فرض

فصل سیزدهم

۱۹۲

فضای نام و دامنه

۱۹۵

دامنه یا قلمرو یک نام

۱۹۶

دامنه محلی

۱۹۷

متغیر غیر محلی

۱۹۸

دامنه سراسری

۱۹۹

متغیر سراسری

فصل چهاردهم

۲۰۳

شی گرایي در پایتون

۲۰۴

برنامه نویسی شی گرا

۲۰۵

مفهوم کلاس

۲۰۸

تعریف کلاس در پایتون

۲۱۱

سازنده یک کلاس

۲۱۳

صفت ها

۲۱۶

شگردهای یک کلاس

۲۲۱

دستکاری اشیاء

۲۲۲

ارث بری در پایتون

۲۲۶

ارث بری از چند کلاس

فصل پانزدهم

مدیریت خطا

۲۳۱

۲۳۴

به دام انداختن نوع خاصی از استثنا

۲۳۷

دستور raise

۲۳۸

ایجاد یک استثنای سفارشی

فصل شانزدهم

فایل ها

۲۳۹

۲۴۱

تابع open()

۲۴۳

خواندن فایل

۲۴۳

بستن فایل

۲۴۶

خواندن خط به خط فایل های متنی

۲۴۸

نوشتن در فایل

۲۵۰

بستن خودکار فایل ها

۲۵۱

بکارگیری رهنمون try

۲۵۳

بکارگیری رهنمون with

دیباچه

دوران ما دوران رشد و شکوفایی فن آوری^۱ است. عصری است که در آن فن آوری در تمامی جنبه های زندگی آدمی نفوذ کرده و همه گستره زندگی بشر را به تسخیر خود درآورده است، چنانکه عنوان عصر طلایی فن آوری را برازنده خود ساخته است. از این رو زیستن و بالیدن در روزگار ما در گرو یادگیری دو مهارت اساسی است: مهارت بکارگیری فن آوری و مهارت چیرگی بر فن آوری و مهار آن در جهت اهداف فردی و کاری، با داشتن مهارت نخست شما تنها یک مصرف کننده فن آوری هستید اما با بدست آوردن مهارت دوم شما سوار بر اسب سرکش فن آوری شده و می توانید نیروی مهارناپذیر آن را در دستان خود بگیرید و از توانایی و قابلیت های آن برای آفرینش چیزهای جدید و یا چیرگی بر چالش های فرا روی زندگی کاری و فردی خود بهره گیرید.

بخش بزرگی از فن آوری های عصر ما پیوندی ناگسستنی و جدایی ناپذیر با فن آوری محاسبات و پردازش اطلاعات دیجیتال دارند بگونه ای که رد پای رایانه ها به عنوان مهمترین ابزار محاسبات و پردازش اطلاعات در هر فن آوری دیگری آشکار و هویداست. از این رو می توان گفت که برنامه نویسی یکی از مهم ترین مهارت های مورد نیاز در عصر ماست.

برنامه نویسی هنر فرمانروایی بر دنیای رایانه هاست، شما با داشتن این مهارت پیشرفته و بسیار سودمند می توانید قدرت رایانه ها را برای حل مشکلات و چالش های دنیای واقعی بکار گیرید. برنامه نویسی هنر آفریدن یک برنامه است، آفرینش چیزی که وجود ندارد اما هنگامی که هستی می گیرد، رایانه را و او می دارد تا داده ها و اطلاعات را بر اساس الگوریتم تعیین شده توسط برنامه نویس پردازش کند و محاسبات لازم را بر مبنای آن انجام دهد. و نتیجه تولید شده را دو دستی پیشکش کند. افزون بر این زمانی که شما برنامه نویسی یاد می گیرید، ناگزیرید که منطقی فکر کنید و تفکر منظم و ساخت یافته ای داشته باشید. همچنین مهارت برنامه نویسی هوش و خلاقیت شما را افزایش می دهد و چه دست آوردی بهتر از مهارتی است که می تواند در تمام جنبه های زندگی فردی و کاری شما را یاری دهد.

¹ Technology

این کتاب بر آموزش مهارت برنامه نویسی بر اساس زبان پایتون متمرکز است. پایتون در اواخر دهه ۱۹۸۰ میلادی در موسسه ملی تحقیقات ریاضی و رایانه کشور هلند^۱ پا به عرصه وجود نهاد و به سرعت به یکی از پرتعدادترین و محبوب ترین زبان های برنامه نویسی در جهان تبدیل شد. این زبان همه منظوره که طراح اصلی آن خیدو فان روسوم^۲، برنامه نویس هلندی است به گونه ای طراحی شده است که به جای متمرکز کردن تمامی عملکرد آن در هسته زبان، رویکرد هسته کوچک و افزونه پذیر را در پیش گرفته است از این رو دامنه توانایی ها و قابلیت های آن را می توان با افزودن کتابخانه ها و بسته های نرم افزاری به سادگی گسترش داد. این ویژگی بسیار ارزشمند در کنار سادگی نگارش پایتون و سهولت یادگیری آن موجب شده است که پایتون در دامنه وسیعی از کاربردهای مختلف از هوش مصنوعی و یادگیری ماشینی گرفته تا تجزیه و تحلیل داده ها یا همان علم داده بکار گرفته شود. از این رو تسلط بر این زبان می تواند راهگشای شما در بدست آوردن فرصت های شغلی بسیار باشد.

محمد ایزانلو

^۱ Centrum Wiskunde & Informatica (CWI)

^۲ Guido van Rossum

فصل نخست

نصب پایتون و آماده سازی محیط برنامه نویسی

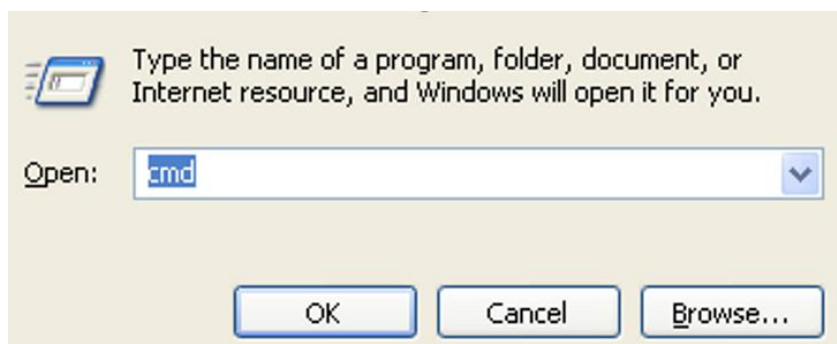
پیش از نصب پایتون با اجرای دستور زیر در محیط خط فرمان ویندوز از نصب بودن آن اطمینان حاصل کنید :

```
C:\Users\Your Name>python -version
```

چنانچه Python در سیستم شما نصب باشد خروجی این دستور نسخه پایتون نصب شده را برای شما نمایش می دهد درغیراین صورت پیام زیرنمایش داده خواهد شد :

Python was not found

برای دسترسی به خط فرمان ویندوز، عبارت cmd را دربخش search منوی start جست و جو کنید ، با ظاهر شدن نتیجه ای به نام command prompt و با کلیک بروی آن می توانید پنجره خط فرمان ویندوز را باز کنید. همچنین می توانید با فشردن همزمان کلیدهای ترکیبی Windows + R در صفحه کلید پنجره ی Run را باز کرده و با نوشتن عبارت cmd و کلیک بروی دکمه ok وارد محیط خط فرمان ویندوز شوید.



تصویر ۱-۱ پنجره run

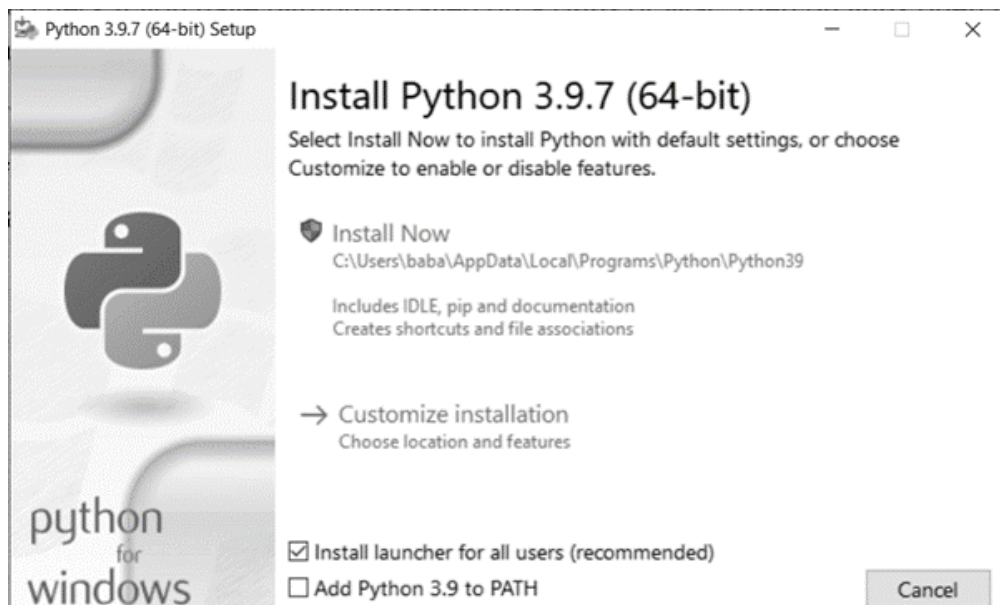
اگر Python درایانه شما نصب نیست می توانید برای دریافت آخرین نسخه آن به بخش Download از تارنمای <https://www.python.org> مراجعه کنید حال بر اساس محیط مورد نظر این آموزش یعنی سیستم عامل Windows و اینکه چه نسخه ای از ویندوز (۳۲ بیتی یا ۶۴ بیتی) بر روی دستگاه شما نصب شده است، نسبت به بارگذاری و دریافت فایل نصب مناسب اقدام کنید لازم به گفتن است که در زمان نگارش این کتاب، امکان انتخاب آخرین نسخه این نرم افزار یعنی ۳,۹,۷ و نسخه قدیمی تر آن یعنی ۲,۷,۱۸ وجود دارد. ما در این آموزش از آخرین نسخه این زبان استفاده خواهیم کرد.

روش نصب

برای نصب Python باید فایل نصب دریافت شده در مرحله قبل را اجرا کنید با اجرای این فایل، برنامه نصب پایتون اجرا شده و پنجره ای نمایش داده می شود که در آن دو روش نصب به شما پیشنهاد می گردد:

۱- نصب پیش فرض

۲- نصب سفارشی



تصویر ۱-۲ پنجره انتخاب نوع نصب (ساده یا سفارشی)

با کلیک بر روی گزینه **Install Now** شما روش نصب پیش فرض پایتون را انتخاب خواهید کرد. در این روش نصب، مسیر نصب پایتون در ویندوز و کتابخانه های مورد نیاز به صورت خودکار توسط نرم افزار نصب، انتخاب خواهند شد و کاربر امکان تغییر مسیر نصب پایتون و یا کم و زیاد کردن کتابخانه ها و بسته های نرم افزاری مورد نظر خود را نخواهد داشت.

با کلیک بر روی گزینه **Customize Installation** نصب پایتون به صورت سفارشی انجام خواهد شد، به این معنی که کاربر می تواند مسیر نصب پایتون را به دلخواه خود تغییر دهد و نیز ویژگی ها و بسته های نرم افزاری مورد نیاز خود را که می خواهد به همراه پایتون نصب شوند انتخاب کند.

در این پنجره با انتخاب گزینه **Install launcher for all user (recommended)** می توانید مجوز اجرای پایتون را به تمامی کاربران تعریف شده در ویندوز اعطاء کنید. انتخاب نشدن این گزینه به معنای این است که تنها کاربر جاری یعنی کاربری که در حال حاضر با آن به ویندوز وارد شده اید می تواند پایتون را اجرا کند. با توجه به اینکه به طور معمول در سیستم عامل ویندوز بیش از یک کاربر تعریف می شود از این رو توصیه می شود این گزینه را انتخاب کنید.

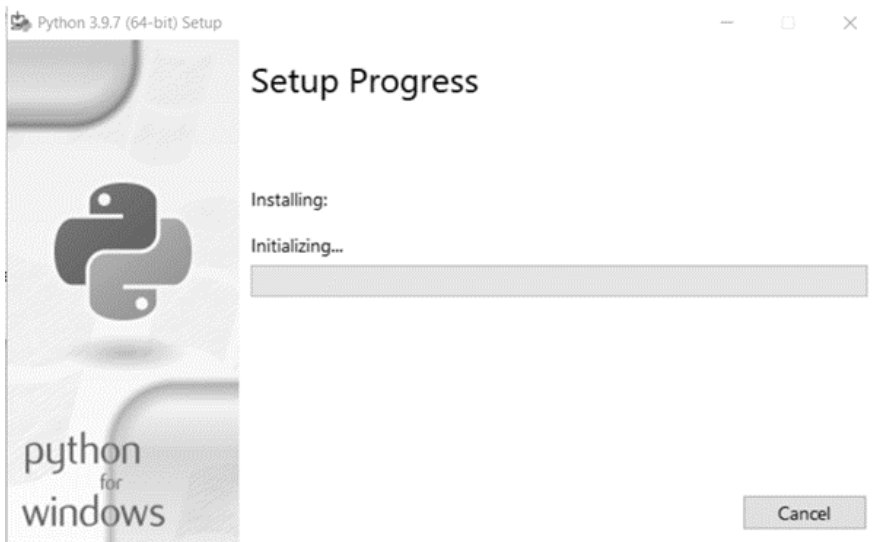
با انتخاب گزینه **Add Python 3.9 to PATH** مسیر نصب پایتون و فایل اجرایی آن به صورت خودکار به متغیر سیستمی **Path** در سیستم عامل ویندوز اضافه خواهد شد لازم به یادآوری است که **Path** یکی از متغیرهای محیطی^۱ سیستم عامل ویندوز است. این متغیر حاوی فهرست پوشه ها و مسیر هایی است که سیستم عامل در آن ها به دنبال یک فایل اجرایی هم نام با دستور وارد شده در خط فرمان می گردد. توصیه میشود این گزینه انتخاب گردد.

چنانچه به هر دلیلی این گزینه را به هنگام نصب پایتون انتخاب نکرده باشید می توانید پس از نصب پایتون و از مسیر **ControlPanel\System and Security\System Properties** به پنجره ویژگی های ویندوز یا همان **SystemProperties** دسترسی پیدا کرده و با کلیک بر روی دکمه

¹ Environment Variables

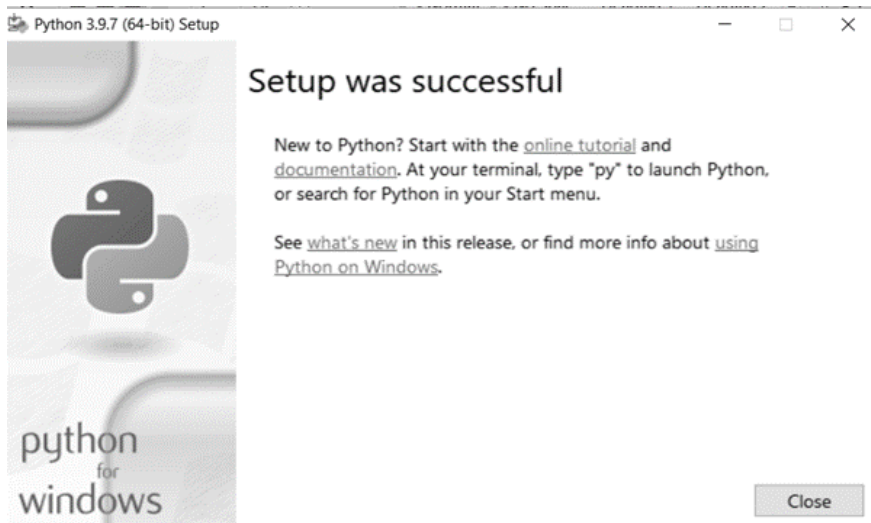
Environment Variables نسبت به اضافه کردن مسیر اجرای Python به متغیر Path موجود در بخش System variable اقدام نمایید.

با کلیک بر روی **Install Now** فرآیند نصب پایتون شروع شده و پنجره زیر نمایش داده می شود.



تصویر ۱-۳ پنجره نمایش پیشرفت فرآیند نصب

و در پایان نمایش پنجره زیر به معنای موفقیت آمیز بودن فرآیند نصب است.



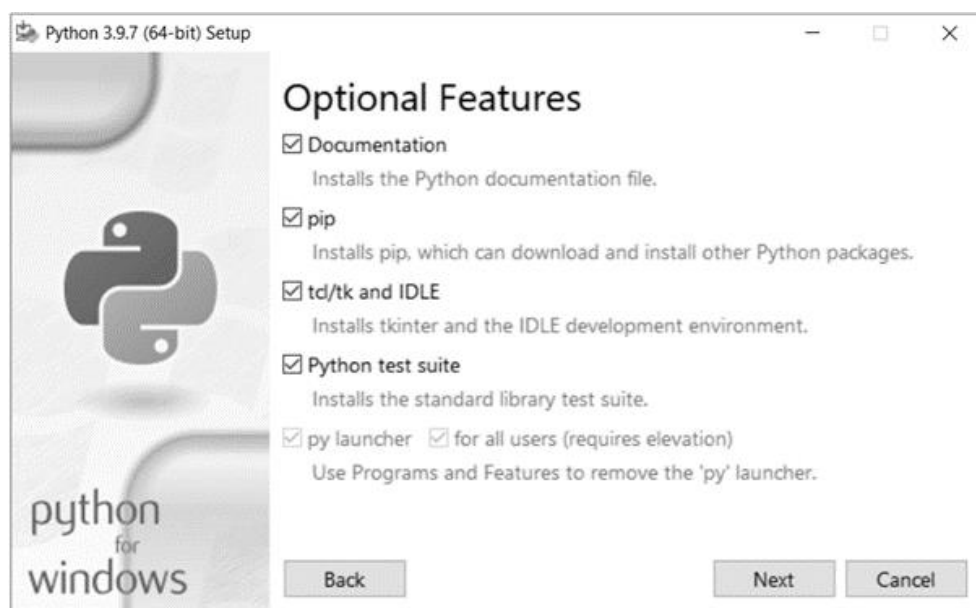
تصویر ۱-۴ پنجره نمایش پایان موفق آمیز فرآیند نصب پایتون

برای اطمینان از نصب صحیح پایتون می توانید دستور زیر را در خط فرمان ویندوز اجرا کنید ،
نمایش نسخه پایتون در خروجی این فرمان گویای موفقیت آمیز بودن نصب پایتون است.

C:\Users\Your Name>python -version

نصب سفارشی

اگر در پنجره اولیه نصب پایتون به جای کلیک بر روی گزینه **Install Now** بر روی گزینه **Customize Installation** کلیک کنید دو پنجره زیر به ترتیب توسط نرم افزار نصب نمایش داده خواهند شد و شما می توانید بسته های نرم افزاری مورد نیاز خود را برای نصب به همراه پایتون انتخاب کنید و مسیر نصب پایتون را به دلخواه خود تغییر دهید همچنین می توانید ویژگی های پیشرفته مورد نظر خود که باید در طول نصب پایتون اعمال گردند را انتخاب کنید و یا از حالت انتخاب خارج سازید.



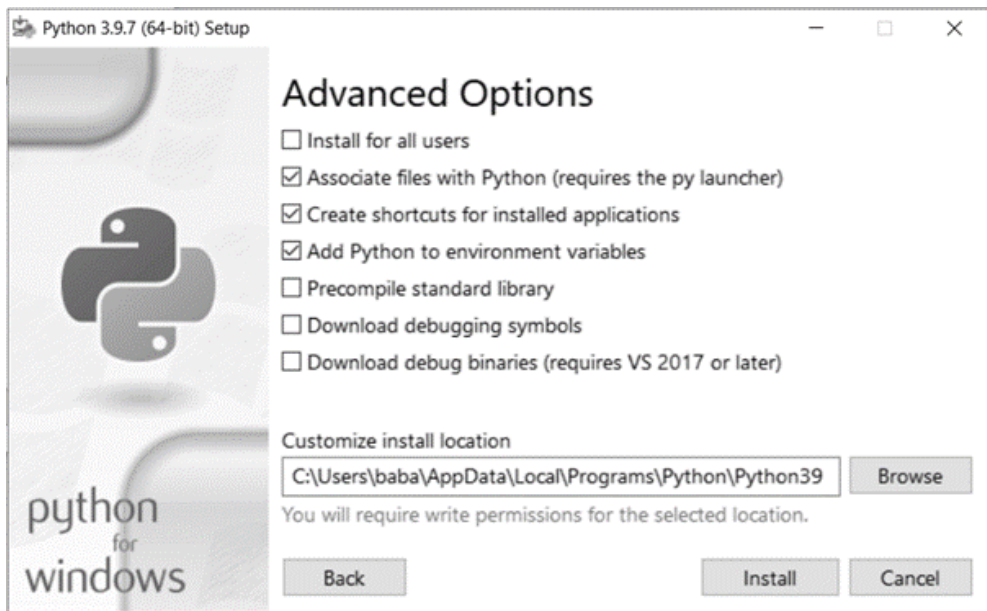
تصویر ۱-۵ پنجره نخست نمایش گزینه های اضافی برای نصب

در این پنجره با انتخاب گزینه Documentaion راهنمای پایتون به همراه دیگر مستندات مورد نیاز در سیستم شما نصب خواهند شد. بهتر است گزینه pip در حالت انتخاب شده باقی بماند pip یک مدیر بسته های نرم افزاری پایتون است و با استفاده از آن می توانید بسته های نرم افزاری یا کتابخانه های مورد نیاز خود را در محیط خط فرمان ویندوز نصب ، حذف و بروزرسانی کنید. برای مثال برای نصب کتابخانه pandas که ویژه تحلیل داده ها در پایتون است از دستور زیر استفاده می شود :

C:\Users\Your Name> pip install pandas

و برای حذف آن می توانید دستور زیر را در خط فرمان ویندوز صادر کنید :

C:\Users\Your Name> pip uninstall pandas



تصویر ۱-۶ پنجره دوم نمایش گزینه های اضافی برای نصب

در پنجره دوم گزینه Associate files with Python فایل های با پسوند .py را به پایتون مرتبط می کند به این معنی که اگر کاربری بر روی فایل با پسوند py کلیک کند این فایل به صورت خودکار توسط پایتون اجرا خواهد شد.

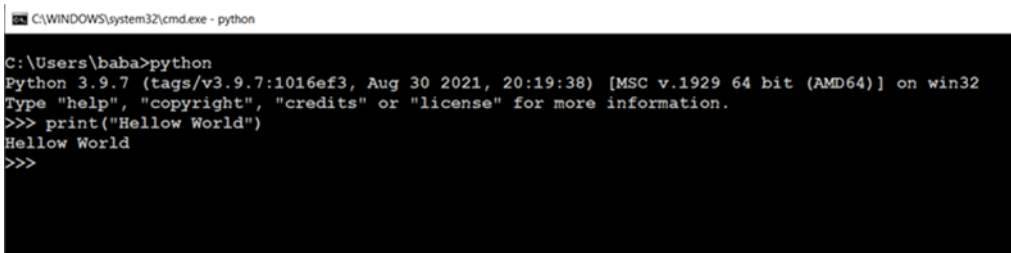
در این پنجره گزینه Add Python to environment variable مسیر نصب پایتون را به فهرست متغیرهای محیطی ویندوز اضافه می‌کند با این کار شما می‌توانید دستور python را بدون اشاره به مسیر کامل نصب پایتون در محیط خط فرمان ویندوز اجرا کنید.

محیط برنامه نویسی و اجرا در Python

۱- برنامه نویسی در خط فرمان پایتون (حالت تعاملی)

با اجرای دستور python یا py در خط فرمان ویندوز می‌توانید به خط فرمان تعاملی مفسر پایتون وارد شوید و برنامه‌های ساده خود را در این محیط بنویسید، اجرا کنید و خروجی آنرا در لحظه مشاهده کنید. با ورود به محیط تعاملی پایتون نشانگر خط فرمان به علامت >>> تغییر خواهد یافت و این به معنای آمادگی مفسر پایتون برای دریافت دستورات زبان برنامه نویسی از کاربر است. در دستورات چند خطی که ادامه دستور در خط و یا خط‌های بعدی قرار خواهد گرفت علامت سه نقطه (...) در ابتدای خط فرمان ظاهر خواهد شد.

این محیط برنامه نویسی مناسب اجرای دستورات ساده پایتون بوده و برای اهداف آموزشی و یا آزمون صحت عملکرد برنامه‌های کوچک پایتون مناسب است. برای خروج از محیط تعاملی برنامه نویسی در پایتون کافی است دستور exit() را اجرا کنید.



```
C:\WINDOWS\system32\cmd.exe - python
C:\Users\baba>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> print("Hello World")
Hello World
>>>
```

در مثال زیر ساده‌ترین دستور پایتون یعنی print("Hello World") که پیام Hello World را به کاربر نمایش می‌دهد، در این محیط اجرا شده است در مثال دوم از دستور if استفاده شده است :

```
Select C:\WINDOWS\system32\cmd.exe - python
C:\Users\baba>python
Python 3.9.7 (tags/v3.9.7:1016ef3, Aug 30 2021, 20:19:38) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>> x=10
>>> if x>5 :
...     print("x bigger than 5")
...     print("ok")
...
x bigger than 5
ok
>>> _
```

۲- ایجاد فایل‌های حاوی دستورات Python

فایل متنی ساده ای که تعاریف و دستورات زبان برنامه نویسی پایتون در آن ذخیره می‌شود را کد منبع^۱، Script و یا ماژول^۲ می‌نامند. ناگفته نماند که تفاوت ظریفی بین یک script و یک ماژول وجود دارد که در آینده و با افزایش دانش و مهارت شما در برنامه نویسی پایتون با آن آشنا خواهید شد اما در این لحظه هر دو را فایل‌هایی با پسوند **.py** می‌دانیم که در بردارنده کدهای پایتون هستند.

برای ساخت یک Script نیاز به هیچ ابزار ویژه ای نیست، تنها کافی است با استفاده از یک نرم افزار ساده ویرایشگر متن مانند برنامه Notepad موجود در سیستم عامل ویندوز و یا نرم افزار رایگان Notepad++ یک فایل متنی ساده با پسوند **.py** ایجاد کنید، دستورات و تعاریف مورد نیاز برنامه خود را در آن وارد نمایید و سپس فایل ساخته شده را ذخیره کنید. ما در سرتاسر این کتاب کدهای خود را با نرم افزار Notepad++ خواهیم نوشت. این نرم افزار رایگان را می‌توانید از نشانی <https://notepad-plus-plus.org> دریافت کنید.

برای اجرای Script نوشته شده تنها کفایت در خط فرمان سیستم عامل ویندوز دستور فراخوانی مفسر پایتون یعنی **python** و یا **py** به همراه نام و مسیر کامل کامل script مورد نظر اجرا گردد. برای مثال ما با استفاده از نرم افزار Notepad++ یک اسکریپت ساده که تنها حاوی یک دستور ساده (**print("Hello World")** است را ساخته و آن را با نام **firstprog** و با پسوند **.py** در مسیر **D:\test** ذخیره کرده ایم بنابراین مسیر کامل دسترسی به این اسکریپت عبارت است از :

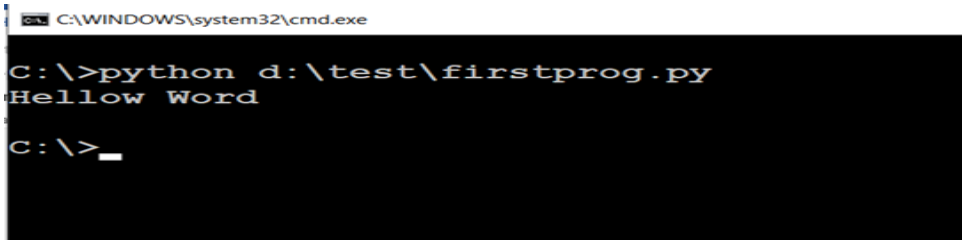
¹ Source Code

² Module

D:\test\ firstprog.py

حال برای اجرای این اسکریپت کافی است تا دستور زیر را در خط فرمان ویندوز اجرا کنید :

C:\> python d:\test\firstprog.py

A screenshot of a Windows command prompt window. The title bar at the top reads 'C:\WINDOWS\system32\cmd.exe'. The command prompt shows the command 'C:\>python d:\test\firstprog.py' being entered, followed by the output 'Hellow Word' (note the typo). The prompt then returns to 'C:\>_'.

```
C:\WINDOWS\system32\cmd.exe
C:\>python d:\test\firstprog.py
Hellow Word
C:\>_
```

۳- محیط توسعه یکپارچه^۱

یک محیط توسعه یکپارچه (IDE) نرم‌افزاری است که امکاناتی چون : ایجاد و مدیریت یک پروژه برنامه نویسی ، ویرایشگر کد ، مفسر^۲ و یا کامپایلر^۳ را در کنار قابلیت‌های مهمی چون اشکال زدایی از برنامه ، تکمیل‌کننده هوشمند کد و دیگر امکانات مفید و سودمند را یکجا و در کنار هم در اختیار برنامه نویس قرار می‌دهد. به ساده‌ترین و رساترین بیان ممکن محیط توسعه یکپارچه ، هر آنچه را که یک برنامه نویس جهت نوشتن ، اشکال زدایی و اجرای کد در کمترین زمان ممکن نیاز دارد را در یک نرم‌افزار واحد یکپارچه کرده و به صورت یکجا در اختیار برنامه نویس قرار می‌دهد.

برای زبان برنامه نویسی Python محیط‌های توسعه یکپارچه گوناگونی وجود دارد که یکی از محبوب‌ترین نمونه‌های آن نرم‌افزار pycharm است که توسط شرکت jetbrains توسعه داده شده و پشتیبانی می‌شود. PyCharm IDE یکی از معمول‌ترین محیط‌های توسعه است که توسط توسعه‌دهندگان و برنامه‌نویسان حرفه‌ای Python استفاده می‌شود. هرچند که این محیط می‌تواند به برنامه‌نویسان تازه‌کار نیز کمک کند تا کدهای خود را بهتر و با سرعت بیشتری تولید کنند. این محیط برنامه نویسی پر طرفدار را می‌توانید از نشانی زیر دریافت کنید.

¹ Integrated Development Environment

² Interpreter

³ Compiler

<https://www.jetbrains.com/pycharm/>

به همراه خود پایتون یک IDE ساده به نام ^۱ IDLE نصب می شود که ساده ترین ویرایشگر کد برای پایتون است و می توان برای اهداف آموزشی و یادگیری از آن استفاده کرد.

از دیگر محیط های توسعه یکپارچه محبوب و پرکاربرد پایتون می توان به دو محیط زیر اشاره کرد :

۱- Visual Studio Code محصول قدرتمند و رایگان شرکت Microsoft

۲- Anaconda Navigator یک محیط گردآورنده است به این معنی که خود آن یک

IDE نیست اما دربردارنده مجموعه ای کامل از نرم افزارهای حرفه ای کدنویسی با پایتون

از جمله تعدادی IDE مناسب مانند pycharm ، spyder و vs است.

¹ Integrated Development and Learning Environment

فصل دوم

آشنایی با مفهوم متغیر و دیگر واژگان دنیای برنامه نویسی

کد ماشین یا زبان ماشین^۱ یک مجموعه از دستورات در قالب دودویی (رشته ای از صفر و یک ها) است که واحد پردازش مرکزی^۲ یک رایانه می تواند آنرا به طور مستقیم اجرا کند. هر دستور یک کار خاص را انجام می دهد، برای مثال: بارگذاری داده از حافظه به CPU و یا انجام یک عمل ریاضی بر روی یک واحد از داده ها. زبان ماشین یک زبان سطح پایین نامیده می شود چرا که به طور مستقیم و بی واسطه توسط واحد پردازش مرکزی رایانه اجرا می شود. همچنین درک و بخاطر سپاری دستورات آن برای انسان بسیار سخت و دشوار بوده و برنامه نویسی با آن طاقت فرسا است. در مقابل به زبانهایی چون Java، C++ و Python که شیوه نگارش آنها به زبان انگلیسی نزدیک بوده و درک دستورات و منطق مدیریت داده ها در آن برای آدمی آسان است زبانهای سطح بالا گفته می شود.

هنگامی که برنامه ای را با یک زبان برنامه نویسی سطح بالا می نویسیم، رایانه و یا بهتر بگوییم واحد پردازش مرکزی توانایی اجرای مستقیم برنامه مارا نخواهد داشت چرا که تنها زبان قابل اجرا توسط رایانه، زبان ماشین است، از این رو ما نیاز به ابزاری داریم که بتواند کد نوشته شده با یک زبان سطح بالا را به کد زبان ماشین تبدیل کند. این نرم افزار واسط که وظیفه تبدیل کد تولید شده توسط زبان برنامه نویسی سطح بالا به کد زبان ماشین را بر عهده دارد بر اساس چگونگی رفتار آن با کد زبان سطح بالا به نام های همگردان^۳ و یا مفسر^۴ شناخته می شود. به بیان دیگر تفاوت، در روش خواندن و زمان ترجمه کدهای زبان سطح بالا به زبان ماشین است، همگردان در ابتدا تمامی

¹ Machine code

² Central processing unit

³ Compiler

⁴ Interpreter

کد سطح بالا را می خواند و همه آن را به یکباره به کد ماشین قابل اجرا در سخت افزار تبدیل می کند. در حالی که مفسر، کد سطح بالا را خط به خط می خواند و هر خط را جداگانه به زبان ماشین ترجمه و اجرا می کند. به زبان ساده تر در روش کامپایلری کد ابتدا ترجمه و سپس اجرا می شود اما در روش مفسری کد خط به خط ترجمه و اجرا می گردد. و ترجمه هر خط و اجرای آن همزمان است. در این جاست که دسته بندی جدیدی از زبانهای سطح بالا شکل می گیرد :

۱- **زبانهای مفسری** که برای ترجمه کد تولید شده توسط زبان سطح بالا به کد ماشین از مفسر استفاده می کنند. Python، PHP و Java Script از شناخته شده ترین زبانهای برنامه نویسی مفسری هستند.

۲- **زبانهای همگردانی** که برای ترجمه کد تولید شده توسط زبان سطح بالا به کد ماشین از همگردان^۱ استفاده می کنند . زبانهای C++ ، C# و Java از شناخته شده ترین زبانهای برنامه نویسی کامپایلری هستند.

محیط توسعه یکپارچه

یک محیط توسعه یکپارچه^۲ نرم افزاری است که امکاناتی چون : ایجاد و مدیریت یک پروژه برنامه نویسی ، ویرایشگر کد ، مفسر و یا کامپایلر را در کنار قابلیت های مهمی چون اشکال زدایی از برنامه ، تکمیل کننده هوشمند کد ، و دیگر امکانات مورد نیاز برنامه نویسی فردی یا گروهی را یکجا و در کنار هم در اختیار برنامه نویس قرار می دهد. به ساده ترین و رساترین بیان ممکن **محیط توسعه یکپارچه** ، هر آنچه را که یک برنامه نویس جهت نوشتن ، اشکال زدایی و اجرای کد در کمترین زمان ممکن نیاز دارد را در یک نرم افزار واحد یکپارچه کرده و به صورت یکجا در اختیار برنامه نویس قرار می دهد.

برای زبان برنامه نویسی Python محیط های توسعه یکپارچه گوناگونی وجود دارد که یکی از محبوب ترین نمونه های آن نرم افزار pycharm است که توسط شرکت JetBrains توسعه داده شده و پشتیبانی می شود. PyCharm IDE یکی از معمول ترین محیطهای توسعه است که توسط

¹ Compiler

² Integrated Development Environment (IDE)

توسعه دهندگان و برنامه نویسان حرفه ای Python استفاده می شود. هرچند که این محیط می تواند به برنامه نویسان تازه کار نیز کمک کند تا کدهای خود را بهتر و با سرعت بیشتری تولید کنند. برای دریافت این محیط برنامه نویسی پر طرفدار را می توانید از نشانی زیر دریافت کنید.

<https://www.jetbrains.com/pycharm/>

از دیگر محیط های توسعه یکپارچه محبوب و پرکاربرد پایتون می توان به دو محیط زیر اشاره کرد :

۱- Visual Studio Code محصول قدرتمند و رایگان شرکت Microsoft

۲- Anaconda Navigator یک محیط گردآورنده است به این معنی که خود آن یک IDE

نیست اما دربردارنده مجموعه ای کامل از نرم افزارهای حرفه ای کدنویسی با پایتون از جمله تعدادی IDE مناسب مانند pycharm ، spyder و vs است.

لازم به یادآوری دوباره است که ما در سرتاسر این کتاب کدهای خود را با نرم افزار Notepad++ خواهیم نوشت. این نرم افزار رایگان را می توانید از نشانی زیر دریافت کنید:

<https://notepad-plus-plus.org>

آشنایی با مفهوم متغیر

فرض کنید می خواهید برنامه ساده ای بنویسید که مساحت یک مثلث را محاسبه و در خروجی چاپ کند. از دیدگاه ریاضی می دانیم که مساحت یک مثلث از رابطه زیر بدست می آید :

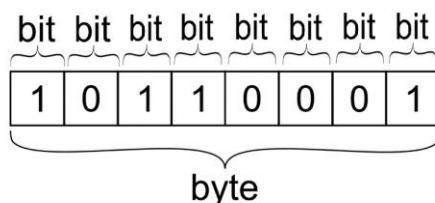
$$\text{مساحت مثلث} = \frac{\text{اندازه قاعده مثلث} \times \text{ارتفاع مثلث}}{2}$$

بنابراین برای محاسبه مساحت یک مثلث ، برنامه ما باید دو مقدار ارتفاع و قاعده مثلث را از ورودی خوانده و سپس آنها را برای انجام محاسبات بعدی در یک مکان مشخص در حافظه اصلی رایانه^۱ ذخیره کند و پس از آن پردازشگر رایانه بر اساس رابطه بالا پردازش لازم برای محاسبه مساحت مثلث را انجام می دهد و نتیجه را به صورت موقت

^۱ RAM

در حافظه اصلی ذخیره می کند و سپس برای نمایش به کاربر ، آنرا به صفحه نمایشگر و یا خروجی چاپگر می فرستد. نیاز است بدانید که RAM را حافظه اصلی می نامند چرا که در برهم کنش و ارتباط سراسر با پردازنده^۱ است و هیچ برنامه ای نمی تواند توسط پردازنده اجرا شود مگر آنکه پیش از آن درون حافظه اصلی نشسته باشد.

در زبان های برنامه نویسی به فضای مشخصی از حافظه اصلی رایانه که برای ذخیره سازی موقت داده های مورد نیاز برنامه و یا نتایج حاصل از پردازش داده ها تخصیص می یابد متغیر^۲ گفته می شود. به زبان ساده تر متغیر فضایی از حافظه اصلی رایانه است که برای ذخیره موقت داده ها در طول اجرای برنامه کنار گذاشته می شود. از این دیدگاه ، ساده تر آن است که متغیر را ظرفی در نظر بگیرید که به ما اجازه می دهد تا داده های خود را درون آن ذخیره کنیم و یا در زمان مورد نیاز داده ها را از آن برداشته و بکار گیریم. لازم به گفتن است که مدیریت حافظه یکی از وظایف بسیار مهم سیستم عامل است و کامپایلر و یا مفسر فضای مورد نیاز برای ذخیره داده های مورد نیاز برنامه را از سیستم عامل تقاضا می کنند. کوچکترین واحد اطلاعات قابل پردازش در رایانه ها بیت^۳ نامیده می شود و هر بیت در هر زمان تنها می تواند یکی از دو مقدار ۰ یا ۱ را داشته باشد. هر هشت بیت را یک بایت^۴ گویند.



شکل ۱-۲ هر بایت از ۸ بیت تشکیل شده است

^۱ CPU

^۲ variable

^۳ Bit

^۴ Byte

سیستم عامل ، حافظه اصلی رایانه را دنباله ای از بایت ها می بیند که بصورت خطی پشت سرهم قرار گرفته اند و هریک از آنها دارای یک نشانی^۱ یکتا و یگانه است. به هر بایت در حافظه اصلی یک خانه حافظه گفته می شود. هر خانه حافظه یک بایت گنجایش دارد و می تواند ۸ بیت اطلاعات را در خود جای دهد.

Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000100
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011

شکل ۲-۲ حافظه اصلی رایانه مجموعه ای از بایت هاست که هر بایت دارای یک نشانی یکتا است

همانگونه که در دنیای واقعی برای استفاده از یک ظرف ، نخست باید آنرا ساخت، به آن نامی داد و سپس چیزی متناسب با جنس و ظرفیت ظرف درون آن ریخت و در زمان مورد نیاز ، آنچه درون ظرف است را بکار گرفت ، در زبانهای برنامه نویسی هم تعریف و استفاده از متغیر در سه گام انجام می شود :

در گام نخست متغیر مورد نظر تعریف می شود یعنی به آن یک نام داده می شود و نوع داده ای که می تواند در متغیر ذخیره شود نیز مشخص می گردد در دومین گام به متغیر تعریف شده مقداری

¹Address

اختصاص می یابد یعنی داده ای در آن ذخیره می گردد و آخرین گام دسترسی به متغیر و واکشی مقدار ذخیره شده در آن است.

از این رواز دیدگاه زبان برنامه نویسی یک متغیر با چهارویژگی اصلی شناخته می شود :

۱- متغیر دارای یک نام است که به آن شناسه^۱ گفته می شود اختصاص نام به یک متغیر از رهنمون های ویژه ای پیروی می کند که در ادامه این درس به اصول نامگذاری متغیرها در زبان پایتون اشاره خواهد شد.


۲- متغیر دارای یک نوع^۲ است ، نوع یک متغیر بیانگر نوع داده ای است که می تواند در متغیر ذخیره گردد. برای مثال اگر یک متغیر از نوع رشته ای تعریف گردد دیگر نمی توان در آن یک عدد ذخیره کرد.

تعیین نوع داده بسیار مهم است چرا که هر نوع داده نیازمند پردازش ، در نهایت به دنباله ای از بیت های ۰ و ۱ تبدیل شده و به صورت الگوهای بیتی در خانه های حافظه اصلی ذخیره می گردد بنا براین نوع یک متغیر است که مفسر و یا کامپایلر را یاری می دهد تا تفسیر درستی از بیت های ذخیره شده در حافظه داشته باشد. برای مثال اگر الگوی بیتی ۰۱۰۰۰۰۰۱ در خانه ای از حافظه (برای نمونه متغیری به نام age) ذخیره شده باشد بدون مشخص بودن نوع متغیر مربوطه این الگو هم می تواند به عنوان عدد ۶۵ تفسیر گردد و هم می تواند به عنوان کد اسکی نویسه A در نظر گرفته شود. فایده دیگر تعیین نوع داده برای یک متغیر امکان تعیین پیشینه تعداد بایت های مورد نیاز برای ذخیره مقدار متغیر در حافظه اصلی است.

¹ Identifier

² Type

Address	Value
0x00	01001010
0x01	10111010
0x02	01011111
0x03	00100100
0x04	01000001
0x05	10100000
0x06	01110100
0x07	01101111
0x08	10111011
...	...
0xFE	11011110
0xFF	10111011


 کاراکتر A
 یا
 عدد ۶۵

شکل ۲-۳ تفسیر الگوی بیتی موجود در یک خانه حافظه

پرکاربردترین انواع داده ای در بیشتر زبانهای برنامه نویسی عبارتند از :

۱- داده های عددی شامل اعداد صحیح و اعشاری مانند : ۱۲ ، ۸.۵ ، -۲۵

۲- داده های متنی شامل حرف و رشته مانند : a ، m ، arash

۳- داده های منطقی شامل دو مقدار TRUE (درست) و FALSE (نادرست)

۴- داده های مجموعه ای مانند آرایه ها ، لیست ها ، مجموعه و ...

۵- داده های ساخت یافته مانند کلاسها و ...

۳- مقدار^۱ متغیر که همان داده ذخیره شده در متغیر است . دادن مقدار به یک متغیر می تواند به هنگام تعریف متغیر انجام شود و یا پس از تعریف آن در خط های بعدی کد صورت پذیرد. به عملیات

¹ Value

مقداردهی به یک متغیر انتساب گفته می شود. که در بیشتر زبانهای برنامه نویسی به وسیله عملگر انتساب (=) انجام می گیرد.

۴- دامنه دید و یا حوزه یک متغیر^۱ که بیانگر طول عمر یک متغیر بوده و محدوده ای از کد را مشخص می کند که متغیر در آن معتبر بوده و قابل دسترسی است. در بیشتر زبانهای برنامه نویسی دامنه دید یک متغیر می تواند سراسر کد باشد و یا به بخش خاصی از کد محدود گردد. در حالت اول دامنه دید متغیر را سراسری^۲ و در حالت دوم محلی^۳ می نامند.

تعریف متغیر در Python

تعریف متغیر در پایتون بسیار ساده است ، کافی است تا نامی برای متغیر مورد نظر خود در نظر بگیرید و آنرا مقداردهی کنید ، به محض اختصاص مقدار به یک متغیر ، متغیر ساخته می شود.

مثال :

```
Name = "Arash"
```

```
Pi = 3.14
```

همچنین در پایتون نیازی نیست تا نوع متغیر به هنگام تعریف آن تعیین گردد ، مفسر پایتون نوع یک متغیر را بر اساس مقدار اختصاص داده شده به متغیر تعریف می کند. برای نمونه در مثال بالا متغیر Name از نوع رشته^۴ و متغیر PI از نوع عددی تفسیر خواهد شد. برای درک بهتر این ویژگی پایتون یعنی بی نیازی از تعریف نوع متغیر لازم است بدانید که در بسیاری از زبان های برنامه نویسی شما ناگزیر به تعیین نوع متغیر در زمان تعریف آن هستید برای مثال در C++ برای تعریف یک متغیر با نام Age که می تواند سن یک فرد را در خود نگهدارد و نوع آن یک عدد صحیح است از دستور زیر استفاده می شود :

¹ Variable Scope

² Global

³ Local

⁴ String

`Int Age = 45;`

همان گونه که دیده می شود نوع متغیر (`int`) در ابتدای تعریف آن و پیش از نام متغیر آمده است. در زبان `Visual Basic` تعریف و مقدار دهی همان متغیر به روش زیر انجام می شود :

`Dim Age as integer = 45`

در مثالی دیگر تعریف یک متغیر به نام `StudentName` که می توان نام یک دانش آموز را در آن ذخیره کرد در زبان `c++` و کد برابر آن در زبان `Visual Basic` در زیر آمده است :

`string StudentName = "Arash"`

`Dim StudentName as string = "Arash"`

در زبان پایتون تعریف و مقدار دهی به متغیرهای تعریف شده در دو مثال بالا بسیار ساده تر شده است :

`Age = 45`

`StudentName = "Arash"`

می توان گفت که هر کد تعریف متغیر در پایتون دارای سه بخش : نام متغیر ، عملگر انتساب و مقدار متغیر است. عملگر انتساب در پایتون همان نویسه `=` است که باید بین نام متغیر و مقدار آن قرار گیرد. برای مثال دستور زیر به مفسر پایتون می گوید که متغیری به نام `Pi` را ایجاد و مقدار عددی `۳,۱۴` را در آن ذخیره کند.

`Pi = 3.14`

پس از تعریف یک متغیر و اختصاص مقدار به آن ، باز هم می توانید مقدار آنرا تغییر دهید، در این صورت مفسر پایتون نوع متغیر را بر اساس آخرین مقدار اختصاص داده شده به آن در نظر خواهد گرفت. و مقدار موجود در متغیر نیز آخرین مقدار اختصاص داده شده به آن خواهد بود.

مثال :

`X = 4`

```
X = "Arman"
```

```
Print(x)
```

در مثال بالا در خط اول نوع متغیر X یک عدد صحیح در نظر گرفته خواهد شد و در خط دوم نوع همان متغیر به رشته ای تغییر خواهد یافت. و خروجی دستور print که بر روی نمایشگر برای کاربر نمایش خواهد یافت نیز Arman خواهد بود.

اگر می خواهید چند متغیر را به صورت همزمان با مقداری یکسان مقداردهی کنید، از روش بکاررفته در مثال زیر استفاده کنید :

مثال : در این مثال به چهار متغیر d , c , b , a مقدار یکسان ۵۵ داده شده است. بنابراین تمامی دستورات print عدد ۵۵ را چاپ خواهند کرد.

```
a = b = c = d = 55
```

```
print(a)
```

```
print(b)
```

```
print(c)
```

```
print(d)
```

چنانچه متغیرهایی که می خواهید همزمان مقداردهی کنید دارای مقدار متفاوت هستند می توانید از روش زیر استفاده کنید :

```
name, family, age = "arash", "izanlou", 15
```

در این روش ، مقادیر تمامی متغیرهای سمت چپ عملگر انتساب = به ترتیبی که نامگذاری شده اند مقداردهی می شوند و مقدار هر متغیر با علامت ، از مقدار متغیر پیش از خود جدا می شود.

قوانین و اصول نام گذاری متغیرها در پایتون

۱- نام متغیرها نمی تواند با عدد آغاز شود و باید با یک حرف و یا علامت خط زیر (_) شروع گردد.

۲- نام متغیر نباید همنام دستورات و رهنمون های زبان برنامه نویسی پایتون باشد.

۳- پایتون یک زبان حساس به بزرگی و کوچکی حروف است از این رو دو متغیر همنام که با حروف کوچک و بزرگ متفاوت نامگذاری شده اند رایکسان فرض نمی کند و آنها را دو متغیر جداگانه در نظر خواهد گرفت. برای مثال `age = 25` و `Age = 25` از نظر پایتون دو متغیر جداگانه هستند.

۴- در نام متغیرها تنها می توان از حروف بزرگ و کوچک الفبای زبان انگلیسی ، اعداد و خط زیر (_ , 0-9 , A-Z , a-z) استفاده کرد و بکارگیری دیگر نشانه ها و علائم مجاز نیست. برای مثال استفاده از فضای خالی (Space) در نام گذاری متغیرها مجاز نیست.

چند نمونه از نامگذاری صحیح متغیر ها :

```
myName = "arman"
```

```
my_name = "arman"
```

```
_my_name = "arman"
```

```
MY_name = "arman"
```

چند نمونه از نامگذاری نادرست متغیرها :

```
2mayname = "arman"
```

```
My-name = "arman"
```

```
My name = "arman"
```

۵- با هدف افزایش خوانایی برنامه و درک بهتر کد ، نام متغیر ها را با معنی و فراخور داده ای که در آن ذخیره می شود انتخاب کنید برای نمونه بهتر است متغیری که نام یک دانش آموز را در خود نگهداری می کند `StudentName` باشد تا نامهایی چون `sn` یا `name`.

۶- چنانچه نام متغیر از چند واژه مختلف تشکیل شده باشد خواندن آن دشوار است و موجب کاهش خوانایی برنامه و سخت تر شدن درک کد نوشته شده می شود از این رو بهتر است در نامگذاری چنین متغیرهایی از یکی از سه شیوه زیر در نامگذاری متغیر پیروی کنید :

روش **کوهان شتری**^۱: در این روش بجز نخستین واژه ، دیگر واژه ها با حروف بزرگ آغاز می شوند. یعنی به جز نخستین واژه ، اولین حرف دیگر واژه ها با حرف بزرگ نوشته می شود.

مثال :

```
studentLastName = "arash"
```

روش **Pascal Case** : در این روش هر واژه جداگانه موجود در نام متغیر، با حروف بزرگ آغاز می شود

مثال :

```
StudentLastName = "arash"
```

روش **حالت مار**^۲ : در این روش هر واژه با یک علامت خط زیر (_) از واژه پیش از خود جدا می گردد :

مثال : `student_last_name = "arash"`

¹ camel Case

² snake_case

فصل سوم

مستند سازی کد و انواع داده

فرض کنید برنامه ای نوشته اید که دارای هزار خط کد است و در آن از متغیرها ، دستورات و سایر عناصر اصلی یک زبان برنامه نویسی مانند حلقه ، تابع ، کلاس و دیگر رهنمون ها و ساختارهای برنامه نویسی که در آینده با آنها آشنا خواهید شد استفاده کرده اید و برای دستیابی به اهداف مشتری و حل برخی از مسائل گوناگون دنیای کسب و کار و پیاده سازی منطق تجاری ، الگوریتم های پیچیده ای را طراحی و کد نویسی کرده اید. برنامه شما پس از پایان پروژه و پشت سر گذاشتن چندین آزمون درستی کارکرد و تایید اولیه ، به مشتری تحویل داده می شود و در محیط واقعی و عملیاتی سازمان مشتری مستقر و اجرا می گردد. شش ماه پس از تحویل ، بازخوردهای داده شده از مشتری نشان می دهد که بخشی از برنامه ، خواسته های مشتری را بدرستی برآورده نمی سازد و نیاز به اصلاح و تغییر دارد. شما در این مدت منطق تجاری و فلسفه حاکم بر طراحی و پیاده سازی بسیاری از الگوریتم های کد نویسی شده در برنامه را از یاد برده اید و باید زمان بسیار زیادی را صرف خواندن خط به خط کد و درک مجدد منطق برنامه و یافتن مشکل کنید بدتر از این زمانی است که بنا بر تصمیم مدیران شرکت توسعه دهنده نرم افزار که در آن کار می کنید فرد دیگری مسئول بروزرسانی و اصلاح این کد شود . خدا به چنین فردی رحم کند ، کد شما بدون توضیحات و مستند سازی لازم ، همانند کلاف سردرگمی است که بازکردنش گاو نر می خواهد و مرد کهن .

از این رو برنامه نویسان حرفه ای ، تنها به مستندات گوناگونی که در مراحل مختلف توسعه یک نرم افزار تولید می شوند دل نمی بندند و خود تمامی دستورات ، متغیرها و عملکرد بخشهای مختلف برنامه ای که کد نویسی کرده اند را توضیح داده و مستند می کنند تا فهم آن برای خود برنامه نویس و دیگر افرادی که کد نوشته شده را بررسی یا استفاده خواهند کرد آسان بوده و به روشنی و بدون ابهام درک گردد.

در زبان پایتون برای افزودن توضیحات و یادداشت نویسی در متن کد و مستند سازی آن از علامت `#` استفاده می شود. مفسر پایتون هر نوشته ای که پس از علامت `#` قرار بگیرد را یادداشت برنامه نویس فرض کرده و از اجرای آن سر باز می زند.

مثال :

```
Age = 18 #student's Age
Name = "arash" #student's Name
Family = "Izanlou"
#student's Family
```

همانگونه که در مثال بالا دیده می شود یادداشت نویسی هم می تواند در ادامه یک خط از کد برنامه انجام شود و هم در یک خط جداگانه صورت پذیرد.

توضیحات چند خطی در پایتون

لازم است بدانید که در پایتون علامت `#` برای نوشتن یادداشت های تک خطی است و در این زبان روش خاصی برای درج توضیحات چند خطی در کد برنامه وجود ندارد اما به این منظور می توانید از دو روش زیر استفاده کنید :

روش نخست : قرار دادن هر یادداشت در یک خط جداگانه

مثال :

```
#This is the First line of Comment
# This is the Second line of Comment
print("Hello World")
```

روش دوم : قرار دادن متن چند خطی مورد نظر در بین دو علامت `"""` یا نقل قول سه گانه^۱، دلیل بهره گیری از این روش، نادیده گرفته شدن رشته های اختصاص داده نشده به یک متغیر توسط مفسر پایتون است.

¹ Triple Quotes

مثال :

```
"""
```

```
This is a Comment
```

```
Written in
```

```
More than one line
```

```
"""
```

```
print("Hello World")
```

دستور print در پایتون

به کمک دستور `print` می توانید متن و یا هر چیز دیگری را بر روی نمایشگر رایانه نمایش دهید به زبان ساده تر `print` یک دستور خروجی در پایتون است. این دستور بسیار پر استفاده بوده و بکارگیری آن ساده است. با امکانات و دیگر قابلیت های دستور `print` پس از آشنایی با مفهوم تابع ، بیشتر آشنا خواهید شد.

مثال :

```
StudentFirstName = "Arman"
```

```
print(StudentFirstName)
```

همانگونه که دیده می شود برای چاپ یک متغیر در خروجی تنها کافیست تا نام آنرا درون تابع `print()` قرار دهید.

انواع داده در Python

نوع داده^۱ یک مفهوم اساسی در زبان های برنامه نویسی است چرا که بخش بسیار مهمی از هر برنامه ، داده هایی است که در آن بکار گرفته می شوند ، این داده ها می توانند یک یا چند ورودی ساده باشند که از کاربر گرفته می شوند و یا داده هایی باشند که برنامه نویس بسادگی می تواند در کد برنامه قرار دهد یا اینکه حجم بزرگی از داده باشد که در یک فایل و بر روی دیسک سخت رایانه نگهداری می شود و یا از شبکه بارگذاری می شود . به هر حال صرف نظر از اینکه سرچشمه داده های مورد نیاز یک برنامه چیست؟ داده ها باید به صورت موقت در حافظه اصلی ذخیره و توسط

¹ Data Type

پردازشگر مرکزی^۱ پردازش شوند یعنی عملیات ویژه ای بر روی آنها انجام شود تا برونداد دلخواه بدست آید و هدفی که برنامه برای آن نوشته شده است تامین گردد. مفسر و یا کامپایلر زبان برای پردازش داده ها لازم است اطلاعاتی در مورد آنها داشته باشد تا بتواند آنها را بدرستی بخواند ، به درستی تفسیر کند و به شیوه ای درست و بهینه ذخیره و بازیابی نماید برخی از مهمترین اطلاعات مورد نیاز جهت تفسیر صحیح داده ها عبارتند از :

- ۱- ماهیت داده : روشن شدن اینکه داده متنی است ؟ عددی است ؟ و یا ساختار دیگری دارد.
 - ۲- بیشینه میزان فضای حافظه مورد نیاز جهت ذخیره و نگهداری داده درحافظه اصلی^۲ بر حسب بایت
 - ۳- عملگرهای مورد نیاز در کار با داده ها : برای مثال عملگرهای ریاضی مانند ضرب و تقسیم برای داده های عددی قابل پذیرش هستند و برای داده های متنی بی معنی خواهند بود.
 - ۴- دامنه دید و یا حوزه یک متغیر^۳ که بیانگر طول عمر یک متغیر بوده و محدوده ای از کد را مشخص می کند که متغیر در آن معتبر بوده و قابل دسترسی است. در بیشتر زبانهای برنامه نویسی دامنه دید یک متغیر می تواند سراسر کد باشد و یا به بخش خاصی از کد محدود گردد. در حالت اول دامنه دید متغیر را سراسری و در حالت دوم محلی می نامند.
- چنانکه در فصل دوم گفته شد نوع داده مهمترین ویژگی یک متغیر است که تمامی اطلاعات مورد نیاز برای تفسیر صحیح مقدار اختصاص داده شده به متغیر را برای مفسر و یا کامپایلر زبان فراهم می کند.

¹ Cpu

² Ram

³ Scope

نوع داده	نام در پایتون	شرح	مثال
متنی	str	رشته	x = "amir"
عددی	Int	عدد صحیح	x = 15 z = -100
	float	عدد اعشاری	x = 12.5 y = 1.0 z = -18.55
	complex	عدد مختلط	z = 5j
دنباله ای	list	فهرست	x = ["ali", "arash", "arman"]
	tuple	چندتایی	tmytuple = ("apple", "banana", "cherry")
	range	محدوده	
مجموعه ای	Set	مجموعه	myset = {"apple", "banana", "cherry"}
	frozenset		
نگاشتی	dict	واژه نامه	thisdict={"brand": "Ford", "model": "Mustang", "year": 1964}
منطقی	bool	منطقی	a = True b = False
دودویی	bytes	بایت ها	
	bytearray	آرایه ای از بایت ها	
	memoryview		

جدول ۱-۳ انواع داده اصلی موجود در زبان برنامه نویسی پایتون

پیدا کردن نوع داده یک متغیر

برای بدست آوردن نوع یک متغیر از دستور `type()` استفاده می شود.

مثال :

```
Age = 15
```

```
Print(type(Age))
```

خروجی دستور بالا `<class 'int'>` است و به این معناست که نوع داده متغیر `Age` عدد صحیح است

مثال :

```
FirstName = "arash"
```

```
Print(type(FirstName))
```

خروجی دستور بالا `<class 'str'>` است و به این معناست که نوع داده متغیر `FirstName` رشته است.

تعیین نوع داده یک متغیر در Python

در پایتون نیازی نیست تا نوع متغیر به هنگام تعریف آن تعیین گردد ، مفسر پایتون نوع یک متغیر را بر اساس مقدار اختصاص داده شده به متغیر تفسیر می کند. برای نمونه در مثال زیر متغیر `StudentFirstName` از نوع رشته ای (`str`) و متغیر `PI` از نوع عدد اعشاری (`float`) تفسیر خواهند شد.

```
StudentFirstName = "Arash"
```

```
Pi = 3.14
```

```
Print(type(StudentFirstName))
```

```
#print <class 'str'>
```

```
Print(type(Pi))
```

```
#print<class 'float'>
```

به هر حال چنانچه می خواهید نوع داده یک متغیر را به هنگام تعریف و مقداردهی آن به صورت صریح مشخص کنید می توانید این کار را همانند مثال های زیر انجام دهید :

```
student_first_name = str("ali")
```

```
student_age = int(25)
```

```
student_weight = float(68.5)
```

```
student_grades = list((17,15,16,19,14,14.5))
```

```
student_is_Single = bool(1)
```

نوع داده عددی

در زبان برنامه نویسی پایتون سه نوع داده عددی وجود دارد که در سه گروه اعداد صحیح (int) ، اعداد اعشاری (float) و اعداد مختلط (complex) دسته بندی می شوند :

مثال :

```
X = 1 # int
```

```
y = 2.8 # float
```

```
z = 5j # complex
```

اعداد صحیح (int) تمامی اعداد بدون اعشار با هر طولی را در بر می گیرد و می تواند علامت دار (منفی یا مثبت) و یا بدون علامت باشد.

مثال :

```
X = 1
```

```
Y = 45685216584985377189
```

```
Z = -32658
```

اعداد اعشاری (float) که به آنها اعداد ممیز شناور و یا اعداد حقیقی نیز گفته می شود ، تمامی اعداد علامت دار (منفی یا مثبت) و یا بدون علامت که دارای یک یا تعداد بیشتری اعشار باشند را در بر می گیرد

مثال :

$$X = 1.15$$

$$Y = 5.0$$

$$Z = -23.18$$

برای مقدار دهی به این دسته از متغیر ها می توان از روش نمادگذاری علمی استفاده کرد.

مثال :

$$X = 35e3$$

$$Y = 12E4$$

$$Z = -87.7e100$$

لازم به گفتن است که نماد گذاری علمی یکی از شیوه های نمایش اعداد بسیار بزرگ و یا بسیار کوچک است که نمی توان آنها را به سادگی در نماد دهی نوشت. در نمادگذاری علمی ، کلیه اعداد به شکل زیر نوشته می شوند :

$$a \times 10^b$$

که در آن b یک عدد صحیح و ضریب a یک عدد حقیقی است.

مثال :

عدد 4000 در نماد گذاری علمی بصورت 4×10^3 نمایش داده می شود و یا عدد -53000 به شکل -5.3×10^4 نمایش داده می شود.

چند مثال بیشتر :

عدد ۶۷۲۰۰۰۰۰۰۰ در نماد ده دهی برابر است با $۶,۷۲ \times ۱۰^9$ در نماد علمی

عدد ۰,۰۰۰۰۰۰۰۰۷۵۱ در نماد ده دهی برابر است با $۷,۵۱ \times ۱۰^{-۹}$ در نماد علمی

برای نمایش نماد گذاری علمی در ماشین حساب ها ، برنامه نویسی و سایر موارد که نمی توان اعداد علمی را به صورت $a \times 10^b$ نشان داد حرف e و یا E جایگزین عدد ۱۰ شده و عدد صحیح b پس از E قرار خواهد گرفت .

مثال : $35e3 = ۳۵۰۰۰$

مثال : عدد ۲۵۰۰۰۰۰ را می توان به صورت $2.5E6$ یا $2.5E + 6$ نوشت

مثال : عدد ۶۰۰۰۰۰ را می توان به صورت $6E5$ یا $6E+5$ نوشت.

اعداد مختلط^۱ : عدد مختلط یک مفهوم ریاضی است که تعریف ویژه خود را دارد، اعداد مختلط دسته ویژه ای از اعداد هستند که از ترکیب یک عدد حقیقی و یک عدد موهومی به دست می آیند. در زبان پایتون ، بخش موهومی اعداد مختلط با پسوند j نمایش داده می شود.

مثال :

$$X = 3+5j$$

$$Y = 8j$$

$$Z = -4j$$

نوع داده منطقی

نوع داده منطقی^۲ تنها می تواند دو مقدار را درون خود نگهداری کند این دو مقدار عبارتند از : درست (True) و نادرست (False)

¹ complex

² Boolean

رشته ها در پایتون

رشته ، دنباله ای از حروف الفبا و یا سایر نویسه های موجود در یک زبان طبیعی مانند انگلیسی و یا پارسی است که به آن متن نیز گفته می شود. این تعریف برای رشته در زبان برنامه نویسی پایتون هم صادق است. نویسه واژه ای پارسی و برابر واژه لاتین character می باشد که کوچکترین واحد متن در یک زبان است. برای نمونه یک حرف انگلیسی ، یک نماد ریاضی مانند + ، یک نشانه نقطه گذاری مانند ؟ !

پردازشگر مرکزی^۱ توانایی پردازش مستقیم حروف و نشانه های یک زبان طبیعی را ندارد و تمام آنچه را که می تواند پردازش کند اعداد دودویی است که دنباله ای از صفر ها و یک ها هستند. به زبان ساده تر داده های ورودی به رایانه از هر نوعی که باشند و با هر ابزاری که به سیستم وارد شوند خواه متنی و طریق صفحه کلید ، خواه صوتی و با میکروفون و خواه تصویری و از طریق اسکنر باید به اعداد دودویی تبدیل شوند تا پردازشگر مرکزی بتواند عملیات لازم را بر روی آنها انجام دهد و نتیجه تولید شده دوباره از قالب دودویی به شکل قابل درک توسط انسان تبدیل و از طریق ابزار خروجی مناسب مانند صفحه نمایش ، چاپگر ، بلندگو و ... در اختیار کاربر قرار گیرد. برای مثال با فشردن کلید a در صفحه کلید ، مدارات الکترونیک موجود در صفحه کلید ، کلید فشرده شده را ابتدا به عدد ۹۷ نگاشت می کند و سپس عدد ۹۷ را به عدد دودویی ۰۱۱۰۰۰۰۱ تبدیل کرده و دست آخر آنچه به پردازشگر مرکزی فرستاده خواهد شده عدد ۰۱۱۰۰۰۰۱ است. بنابراین برای پردازش حروف و الفبای یک زبان طبیعی مانند زبان انگلیسی نیاز است تا تمامی نشانه های زبان طبیعی که به هریک از آنها نویسه^۲ گفته می شود بر اساس یک استاندارد جهانی به مجموعه ای از اعداد نگاشت گردد. به فرآیند نگاشت یک نویسه به یک عدد رمزگذاری^۳ و به فرآیند عکس آن رمزگشایی^۴ گفته می شود.

^۱CPU

^۲ Character

^۳ encoding

^۴ decoding

در حال حاضر دو استاندارد جهانی فراگیر و بسیار پر استفاده برای رمزگذاری نویسه ها وجود دارد :

استاندارد آمریکایی برای تبادل اطلاعات^۱ (Ascii)

استاندارد جهانی^۲ (Unicode)

استاندارد آمریکایی برای تبادل اطلاعات

این استاندارد توسط موسسه استانداردهای ملی آمریکا^۳ تدوین شده است. این استاندارد دیرپاست اما همچنان بصورت گسترده ای برای نمایش و کار با متن به ویژه متن های زبان انگلیسی در رایانه ها بکار گرفته می شود. این استاندارد دو نسخه مختلف دارد که اولی به اسکی استاندارد^۴ معروف است و در آن از هفت بیت برای رمزگذاری هر نویسه استفاده می شود از این رو بیشترین تعداد نویسه ای که می توان با این روش رمزگذاری کرد دو به توان هفت (2⁷) یا ۱۲۸ نویسه است. این روش ۱۲۸ نویسه موجود در زبان انگلیسی را به اعداد ۰ تا ۱۲۷ نگاشت می کند. اسکی استاندارد در قالب یک جدول که به آن جدول اسکی گفته می شود ساده سازی شده است این جدول روشی سودمند برای نمایش نگاشت هر نویسه به اعداد موجود در بازه ۰ تا ۱۲۷ است. بکارگیری این جدول ساده ترین راه برای یافتن کد اسکی هر نویسه است.

نوع دیگر این استانداردها حالت ۸ بیتی است که با نام اسکی توسعه یافته^۵ شناخته می شود در این روش از هشت بیت برای رمزگذاری نویسه ها استفاده می گردد بنابراین توانایی رمزگذاری ۲۵۶ نویسه را دارد و می تواند ۲۵۶ نویسه مختلف را به اعداد ۰ تا ۲۵۵ نگاشت کند. حالت توسعه یافته سازگار با حالت استاندارد است به این معنی که ۱۲۸ نویسه اول که به اعداد ۰ تا ۱۲۷ نگاشت شده اند همانند حالت استاندارد است و نویسه های اضافی پس از آن که از ۱۲۸ تا ۲۵۵ را در بر می گیرد آزاد بوده و می تواند بنا بر نیاز به هر نویسه ای نگاشت گردد. برای مثال در روزگاری که سیستم

¹ American Standard Code for Information Interchange

² Universal Coded Character Set Transformation Format

³ American National Standards Institute (ANSI)

⁴ Standard Ascii

⁵ Extended Ascii

عامل بسیار قدیمی DOS فرمانروای بی رقیب دنیای سیستم عامل های ویژه رایانه های شخصی بود و از زبان فارسی هم پشتیبانی نمی کرد برخی از شرکت های ایرانی از این ظرفیت اسکی توسعه یافته برای معرفی نویسه های زبان فارسی به این سیستم عامل استفاده می کردند. برای نمونه رمزگذاری ایران سیستم که توسط شرکت ایران سیستم معرفی شد و در برنامه های مبتنی بر سیستم عامل DOS در ایران کاربرد بسیار گسترده ای داشت.

در هر دو نسخه اسکی هر نویسه یک بایت از حافظه اصلی را مصرف می کند. به همین دلیل است که در زبانی چون C نوع داده ای char که برای ذخیره نویسه های اسکی بکار می رود یک بایت از حافظه را اشغال می کند.

استاندارد Ascii نویسه های نگاشت شده را به دو دسته چایی و غیر چایی افزایش می دهد:

نویسه های چایی به نویسه هایی گفته می شود که توسط انسان قابل مشاهده است و می توان آنها را چاپ کرد. این نویسه ها شامل رقم های 0 تا 9، حروف کوچک a تا z و حروف بزرگ A تا Z و نمادهای نقطه گذاری (نقطه، خط تیره یا خط فاصله، علامت نقل قول، علامت سوال؟ علامت تعجب! و ...) است

نویسه های غیرچایی یا کنترلی به نویسه هایی گفته می شود که نمی توان آنها را چاپ کرد و توسط انسان قابل مشاهده نیستند. از این نویسه ها در رایانه برای اهداف کنترلی استفاده می شود برای مثال نویسه DEL که برای حذف نویسه از یک متن استفاده می شود و یا نویسه CR و LF که در بسیاری از سیستم عامل ها برای مشخص کردن پایان خط در فایل های متنی و شروع خط جدید استفاده می شوند. در استاندارد Ascii تعداد ۳۳ نویسه کنترلی وجود دارند که ۳۲ نویسه اول جدول اسکی (از ۰ تا ۳۱) به همراه نویسه شماره ۱۲۷ (DEL) را در بر می گیرند.

استاندارد جهانی Unicode

بیشینه تعداد نویسه ای که می توان با استاندارد اسکی توسعه یافته کدگذاری کرد ۲۵۶ نویسه است که با در نظر گرفتن ۱۲۸ نویسه اول آن که نویسه های موجود در زبان انگلیسی و لاتین را پوشش می دهد تنها ۱۲۸ عدد دیگر باقی می ماند که می توان برای رمزگذاری نویسه های موجود در دیگر زبانهای زنده دنیا استفاده کرد. این تعداد برای بیشتر این زبان ها بسیار اندک است. همچنین نگاشت

اعداد ۱۲۸ تا ۲۵۵ به نویسه های هر زبان در هر یک از کشورها مستقل از دیگری و بر اساس زبان رسمی آن کشور انجام می شد. برای مثال کد ۱۳۰ در زبان روسی به یک نویسه و در زبان فارسی به نویسه دیگری اختصاص می یافت که این همپوشانی می تواند موجب ناتوانی در خواندن یک متن روسی در محیط نرم افزارهای فارسی گردد و برعکس آن نیز صادق است. به این دلیل نیاز به استاندارد‌ی که بتواند علائم و نشانه های موجود در تمامی زبانهای زنده دنیا را به صورتی یکتا و منحصر بفرد و مستقل از محیط سیستم عامل ، نرم افزار و زبان کدگذاری کند به شدت احساس می شد. این نیاز سرآغاز پیدایش استاندارد جهانی Unicode است. این استاندارد توسط یک انجمن جهانی غیر انتفاعی به نام انجمن یونیکد^۱ ایجاد ، نگهداری و پشتیبانی می شود. این انجمن از گردهم آمدن شرکت های بلند آوازه ای چون Google ، Microsoft ، Oracle ، IBM و دیگر شرکت های بزرگ که رهبری تولید نرم افزار در جهان را در دست دارند تشکیل شده است.

در این استاندارد هر نویسه در هر یک از زبانهای زنده دنیا به یک کد یکتا و مستقل از سیستم عامل و نرم افزار نگاشت می شود که به آن نقطه کد^۲ گفته می شود و در قالب یک عدد مبنای شانزده با پیشوند U+ نمایش داده می شود. برای مثال حرف A بزرگ در زبان انگلیسی به نقطه کد U+0041 نگاشت می گردد که همان عدد ۶۵ در قالب ده دهی است. پیامد مهم این روش ساده اما دشوار یعنی اختصاص یک نقطه کد یکتا به تمامی نویسه های موجود در هر یک از زبانهای زنده دنیا ، امکان بومی سازی و جهانی سازی نرم افزار است و این هدیه ای است که استاندارد یونیکد برای برنامه نویسان و کاربران سرتاسر جهان به ارمغان آورده است .هم اکنون به لطف این استاندارد است که شما می توانید به زبان رسمی کشور خود در صفحات وب و یا در هر نرم افزار دیگری متن مورد نظر خود را بنویسید و مطمئن باشید که دیگران چیزی را که می بینند همان چیزی است که شما نوشته اید. بدون آنکه نگران این باشید که کاربر مقابل از چه سیستم عامل و یا چه مرورگری استفاده می کند. تنها چیزی که لازم است این است که نرم افزار مرورگر و سیستم عامل کاربر از استاندارد یونیکد پشتیبانی کند. در حقیقت استاندارد یونیکد کار دشوار نگاشت هر نویسه در تمامی زبانهای زنده دنیا به یک نقطه کد یکتا را در دستورکار خود قرار داد که پذیرش جهانی این استاندارد و

¹ Unicode Consortium

² CodePoint

کاربرد بسیار گسترده و فراگیر آن در سیستم عامل های پیشرفته امروزی و زبان های برنامه نویسی پرکاربردی چون python, Java و C# نشان دهنده آن است که این استاندارد از این آزمون سخت و دشوار پیروز و سربلند بیرون آمده است.

برای سازگار ماندن یونیکد با استاندارد قدیمی اسکی، تمامی نقطه کدهای U+0000 تا U+007F یعنی از ۰ تا ۱۲۷ به همان کدهای استاندارد اسکی نگاشت شده است.

در استاندارد یونیکد سه روش کدگذاری نویسه وجود دارد :

۱- UTF-8

۲- UTF-16

۳- UTF-32

UTF-8^۱ یک روش رمزگذاری با طول متغیر است و می تواند تا چهار بایت (۳۲ بیت) گسترش یابد به زبان ساده تر هر نویسه در این روش می تواند از یک تا چهاربایت حافظه اشغال کند. این روش تنها از هشت بیت (یک بایت) برای رمزگذاری نویسه های موجود در اسکی استاندارد استفاده می کند. اما برای نمایش دیگر نویسه های موجود در سایر زبانها ممکن است تا چهار بایت هم اختصاص داده شود. برای مثال نویسه های زبان فارسی، عربی و روسی^۲ دو بایتی هستند و بسیاری از نویسه های موجود در زبان های چینی، ژاپنی و کره ای سه بایتی هستند همچنین فضای چهار بایتی برای نمایش شکلک ها^۳ و برخی از نشانه ها و علائم ریاضی استفاده می شود. UTF-8 در حال حاضر پر استفاده ترین روش کدگذاری نویسه در استاندارد Unicode است. دلیل بکارگیری گسترده این روش سازگاری آن با اسکی استاندارد و مصرف کمتر و هوشمندانه حافظه است.

^۱ Unicode Transformation Format 8 bit

^۲ Cyrillic

^۳ Emoji

نگاه فنی تر به Utf-8

می دانیم که فضای مورد نیاز نویسه های رمزگذاری شده در UTF-8 از یک تا چهار بایت متغیر است از این رو نیاز است که تعداد بایت های تخصیص داده شده به یک نویسه قابل تشخیص باشد. یعنی به روشی مشخص گردد که نویسه از یک بایت و یا بیشتر تشکیل شده است. UTF-8 این کار را به شکل زیر و با مقدار دهی به چند بیت اختصاصی در هر بایت انجام می دهد:

- نقطه های کد ۰ تا 007F یعنی ۱۲۸ کاراکتر نخست جدول اسکی استاندارد به صورت معمولی یعنی ASCII تک بایت ذخیره می شوند. از این رو UTF-8، در بایت اول تنها از ۷ بیت آن استفاده می کند و بیت اول آن برای این هدف کنار گذاشته شده است.

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			

جدول ۲-۳ نقطه کدهای تک بایتی

- در نقطه کدهای ۰۰۸۰ و بالاتر بایت نخست نشان دهنده تعداد بایت های بکار رفته در نویسه است که شامل خود بایت نخست هم می شود. در این دسته از نقطه کدها بایت هایی که با الگوی دو بیتی ۱۰ در پرارزشترین مکان بایت آغاز می شوند بایت های داده هستند و شامل اطلاعاتی در مورد نقطه کد هستند. برای مثال نقطه کد دو بایتی دارای الگوی 110xxxxx 10xxxxxx به این معنی است که ۲ بایت در نویسه وجود دارد در این الگو X ها نشان دهنده مقادیر دودویی نقطه کد هستند که باید در یازده بیت باقی مانده ارائه شوند. به طور کلی در این دسته از نقطه کدها تفسیر الگوی قرار گرفتن بیت ها در بایت نخست به صورت زیر است:

110xxxxx : این الگو نشان دهنده وجود ۲ بایت به صورت متوالی در نقطه کد است که شامل بایت نخست هم می شود پس بر اساس جدول زیر ۲ بایت در UTF-8 تنها ۱۱ بیت را ارائه می کند یعنی $2^{11} = 2,048$ نویسه را می توان کد گذاری کرد

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0080	U+07FF	110xxxxx	10xxxxxx		

جدول ۳-۳ نقطه کدهای دو بایتی

1110xxxx : این الگوی موجود در بایت نخست به معنی این است که نقطه کد از ۳ بایت تشکیل شده است پس بر اساس جدول زیر ۳ بایت در UTF-8 تنها ۱۶ بیت را ارائه می‌کند یعنی تعداد $2^{16} = 65,536$ نویسه را می‌توان کد گذاری کرد

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	

جدول ۳-۴ نقطه کدهای سه بایتی

11110xxx : وجود چهار بیت ۱۱۱۱ در پرارزش ترین مکان بایت نخست یعنی ۴ بایت در نقطه کد وجود دارد پس بر اساس جدول زیر ۲۱ بیت کد گذاری تعداد $2^{21} = 2,097,152$ نویسه را فراهم می‌کند.

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

جدول ۳-۵ نقطه کدهای چهار بایتی

First code point	Last code point	Byte 1	Byte 2	Byte 3	Byte 4
U+0000	U+007F	0xxxxxxx			
U+0080	U+07FF	110xxxxx	10xxxxxx		
U+0800	U+FFFF	1110xxxx	10xxxxxx	10xxxxxx	
U+10000	U+10FFFF	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx

جدول ۶-۳ نقطه کدهای Unicode UTF-8

۲- UTF-16

این روش از ۱۶ بیت (دو بایت) برای رمزگذاری یک نویسه استفاده می کند

۳- UTF-32

این روش از ۳۲ بیت (چهار بایت) برای رمزگذاری یک نویسه استفاده می کند

حال که با Unicode آشنا شدید و می دانید که پایتون به صورت پیش فرض از یونیکد پشتیبانی می کند می توان تعریف دقیق تری از رشته ها رو کرد :

رشته در پایتون دنباله ای از نویسه های یونیکد است.

لازم است بدانید که پایتون دارای توانایی بسیار بالایی در کار با رشته ها است که پس از آشنایی بیشتر با این زبان و با استفاده از منابع مختلف می توانید روش های موجود را بیاموزید و یا خود ، ابزارهای مورد نیاز برای کار با رشته ها را پدید آورید.

از دید پایتون هر چیزی که درون یک جفت علامت نقل قول دوتایی " و یا تکی ' قرار گیرد ، رشته تفسیر می شود. برای مثال مقدارهای "۲۵" ، "۶۵،۱۲" و "hello" از دید پایتون رشته هستند بنابراین

برای اختصاص یک مقدار رشته ای به یک متغیر ، مقدار مورد نظر باید درون جفت علامت های " " و یا ' ' قرارگیرد.

مثال :

```
PersonName = "arash"
PersonFamily = 'izanlou'
PersonPhone = "05842221111"
print("Hello World")
print(PersonName)
print(PersonFamily)
print(PersonPhone)
```

برای دادن یک متن چند خطی به یک متغیر باید متن مورد نظر را درون جفت علامت های "" و یا "" قرار دهید.

مثال :

```
PersonActDiscription = "" This is The multi line
String for this example that show you
How to write multi line text ""
PersonActDiscription2 = " This is The multi line
String for this example that show you
How to write multi line text "
Print(PersonActDiscription)
Print(PersonActDiscription2)
```


فصل چهارم

عملگرها

از دوران خوش مدرسه یاد گیری اعمال ریاضی جمع ، تفریق ، ضرب و تقسیم را به یاد دارید ؟ تکلیف سخت حفظ جدول ضرب همراه با قدم زدن در میان برف های سفید و انبوهی که در یک روز سرد زمستانی و در زیر پرتوهای زرین آفتاب در حیاط مدرسه دامن گسترانیده بود را به یاد می آورید؟ چه رویاهای شیرین دوران مدرسه را به یاد داشته باشید و چه از یاد برده باشید نیک می دانید که در درس ریاضی ، ضرب عملیاتی است که با نماد یا علامت \times نمایش داده می شود و عددی که در سمت راست آن قرار دارد (عملوند راست) را به تعداد عدد قرار گرفته در سمت چپ نماد (عملوند چپ) با هم جمع می کند. برای مثال ضرب 3×12 که عملوند راست آن ۱۲ و عملوند چپ آن ۳ است یعنی : $12 + 12 + 12$ پس حاصل 3×12 برابر است با ۳۶ با این تعریف مفهوم دو عبارت ریاضی 3×12 و 12×3 متفاوت از یکدیگر است هرچند که حاصل هر دو عبارت یکی است. عبارت 3×12 یعنی حاصل جمع سه دسته دوازده تایی ($12 + 12 + 12 = 36$) و 12×3 یعنی حاصل جمع دوازده دسته سه تایی. ($3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 + 3 = 36$)

همانند دانش ریاضی در زبانهای برنامه نویسی نیز دو مفهوم عملگر^۱ و عملوند^۲ بسیار پر کاربرد بوده و آشنایی با آنها از نخستین گامهایی است که هنر آموزان برای ورود به دنیای برنامه نویسی برخواهند داشت.

در ساده ترین بیان ، یک عملگر یک نویسه ویژه در زبان برنامه نویسی است که بیانگر یک عملیات مشخص و از پیش تعریف شده در آن زبان است. با بیانی ساده تر می توان گفت که یک عملگر در یک زبان برنامه نویسی علامتی است که انجام عملیات ویژه ای را مشخص می کند. به داده و یا

¹Operator

² Operand

داده هایی که عملیات تعریف شده بر روی آنها انجام می شود عملوند گفته می شود. ترکیبی از یک یا چند متغیر، داده خام، عملگر و عملوند های مربوطه یک عبارت، جمله یا گزاره نامیده می شود.

مثال :

$X=20$ یک عبارت است که از دو عملوند X و 20 و یک عملگر انتساب با نماد $=$ ساخته شده است.

مثال :

$Z+5$ گزاره ای است که از دو عملوند Z و 5 و عملگر ریاضی جمع $+$ تشکیل شده است.

مثال :

$$X = 20$$

$$Y = 40$$

$$Z = x+y$$

در مثال بالا سومین جمله از دو عملگر تشکیل شده است: عملگر انتساب با نماد $=$ و عملگر ریاضی جمع با نماد $+$ ، در این عبارت Z و $X+y$ عملوندهای عملگر انتساب هستند و x ، y ، X عملوندهای عملگر جمع می باشند.

انواع عملگرها در پایتون

پایتون یک زبان همه منظوره است و می تواند در دامنه وسیعی از کاربردهای مختلف از محاسبات علمی و هوش مصنوعی گرفته تا پردازش داده و اطلاعات و ساخت نرم افزارهای کاربردی و تجاری بکار گرفته شود از این رو دارای مجموعه ای غنی از عملگرهای گوناگون است معمول ترین و در عین حال مهم ترین عملگرهای موجود در پایتون عبارتند از:

عملگر انتساب یا تخصیص

این عملگر که با نماد = نمایش داده می شود برای دادن یک مقدار به یک متغیر استفاده می شود به بیان دیگر مقداردهی به متغیر ها توسط این عملگر انجام می شود. در طی عملیات انتساب ، ابتدا عملوند سمت راست عملگر ارزیابی و محاسبه شده و مقدار بدست آمده در عملوند سمت چپ ذخیره می گردد. عملگر انتساب^۱ پرکاربردترین عملگر در زبان های برنامه نویسی است

مثال :

$$X = 20$$

$$Y = 40$$

$$Z = x+y$$

در خط نخست مثال بالا مقدار عددی ۲۰ به متغیر X داده شده است یعنی در متغیر X عدد ۲۰ ذخیره شده است. بنابراین متغیر X حاوی مقدار عددی ۲۰ است.

در خط دوم مقدار عددی ۴۰ به متغیر Y داده شده است یعنی در متغیر Y عدد ۴۰ ذخیره می شود. پس متغیر Y حاوی مقدار عددی ۴۰ است.

در خط سوم ابتدا حاصل عبارت $X+Y$ که مقدار عددی ۶۰ است محاسبه و سپس در متغیر Z ذخیره می شود. از این پس متغیر Z حاوی مقدار عددی ۶۰ خواهد بود.

عملگرهای ریاضی (محاسباتی)

بسیاری از مسائل دنیای واقعی از محاسبه حقوق و دستمزد کارکنان یک شرکت صنعتی گرفته تا محاسبات مورد نیاز برای تعیین مسیر حرکت موشکی که ماهواره ای را به مدار مشخصی از جو زمین منتقل می کند نیازمند پردازش داده های عددی هستند از این رو محاسبات ریاضی یک بخش جدایی ناپذیر از دنیای برنامه نویسی است بنابراین وجود عملگرهایی که بتوانند عملیات ریاضی را در زبان برنامه نویسی پیاده سازی کنند گریزنپذیر خواهد بود. با بکارگیری این عملگرها در زبان

¹ Assignment

های برنامه نویسی می توان عملیات ریاضی ضرب ، تقسیم ، جمع ، تفریق ، به توان رساندن و باقی مانده را انجام داد. برای مثال در بسیاری از زبان های برنامه نویسی عملگر محاسباتی جمع با نماد + نمایش داده می شود این عملگر برای انجام عملیات ریاضی جمع^۱ استفاده می شود این عملگر ، عملوندهای سمت چپ و راست خود را با هم جمع کرده و حاصل عملیات را در اختیار برنامه قرار می دهد. در جدول زیر عملگرهای ریاضی زبان برنامه نویسی پایتون فهرست شده اند .

مثال	نام	عملگر
$x + y$	جمع	+
$x - y$	تفریق	-
$x * y$	ضرب	*
x / y	تقسیم	/
$x \% y$	باقی مانده	%
$x ** y$	به توان رساندن	**
$x // y$	خارج قسمت صحیح	//

جدول ۴-۱ عملگرهای ریاضی در پایتون

مثال ۱:

 $x = 5$
 $y = 3$

```
print(x + y) #output is 8
```

```
print(x - y) #output is 2
```

```
print(x * y) #output is 15
```

¹ Addition

مثال ۲ :

```
x = 12  
y = 3  
print(x / y) #output is 4
```

مثال ۳ :

```
x = 5  
y = 2  
print(x % y) #output is 1
```

مثال ۴ :

```
x = 2  
y = 5  
print(x ** y) #output is ۳۲ same as 2*۲*۲*۲*۲
```

مثال ۵ :

```
x = 15  
y = 2  
print(x // y) #output is 7
```

مثال ۶ :

```
x = 13  
y = 2  
print(x // y) #output is 6
```

عملگرهای مقایسه ای و نوع داده منطقی

مقایسه دو کمیت، فعالیتی معمول در بسیاری از شاخه های دانش است. برای نمونه یک تحلیل گر آماری با مقایسه میانگین قد جمعیت دانش آموزان سال اولی یک مدرسه با میانگین کشوری قد دانش آموزان سال اولی می تواند استدلال کند که میانگین قد دانش آموزان سال اولی این مدرسه از میانگین کشوری قد دانش آموزان سال اولی بیشتر است، کمتر است و یا با آن برابر است. در جریان فعالیت های عادی زندگی هم بسیار پیش می آید که پدیده ها و چیزهای پیرامون خود را از نظر اندازه، بزرگی و کوچکی، بلندی و کوتاهی، قیمت و دیگر ویژگی ها با هم مقایسه کنیم.

زبانهای برنامه نویسی برای حل چالش ها و مسائل پیش روی آدمی در زندگی واقعی طراحی و ساخته شده اند و از آنجا که مقایسه کردن و نتیجه حاصل از آن، مبنای اصلی بسیاری از استدلال های منطقی و تصمیم گیری های هوشمندانه است بنابراین زبانهای برنامه نویسی باید توانایی انجام عملیات مقایسه و بیان نتیجه حاصل را داشته باشند.

در زبانهای برنامه نویسی، دسته ای از عملگرها وجود دارند که نقش اصلی آنها انجام مقایسه بین عملوندها است به این دسته از عملگرها عملگرهای رابطه ای هم گفته می شود. چرا که با انجام مقایسه بین دو مقدار، رابطه بین دو مقدار را به صورت کوچکتر، بزرگتر و یا برابر بیان می کنند.

هر مقایسه دارای سه بخش اساسی زیر است :

۱- دو یا چند چیزی که باید با هم مقایسه شوند برای مثال مقایسه قد دو دانش آموز، مقایسه حجم دو کره و یا مقایسه دو عدد. به هنگام مقایسه دو یا چند پدیده در نظر گرفتن دو نکته لازم و ضروری است: نخست آنکه مقایسه دو چیزی که دارای ماهیت متفاوت هستند بی معنی است برای نمونه نمی توان کمیت قد را با وزن مقایسه کرد و یا فاصله را با سرعت. در برنامه نویسی هم شما نمی توانید یک داده عددی را با یک داده متنی مقایسه کنید. دوم آنکه یکای اندازه گیری دو پدیده ای که دارای ماهیت یکسان هستند باید یکی باشد برای مثال نمی توان دمای اتاقی را که با واحد سانتی گراد اندازه گیری شده با دمای اتاق دیگری که با واحد فارنهایت بیان شده است مقایسه کرد. برای درک بهتر این نکته فرض کنید که پروژه جدید شرکت شما طراحی و پیاده سازی برنامه

یکپارچه حقوق و دستمزد برای یک شرکت چند ملیتی است که کارکنان آن در آمریکای شمالی با واحد پولی دلار حقوق خود را دریافت می کنند در حالی که به کارکنان شرکت که در کشور ژاپن مستقر هستند به واحد پولی ین حقوق پرداخت می شود. بنابراین و با توجه به شناور بودن نرخ برابری ارزهای مختلف ، مقایسه حقوق ماهانه کارمندی که در ژاپن خدمت می کند با حقوق ماهانه کارمند هم تراز وی در آمریکای شمالی بی معنی و فاقد اعتبار است. پس اگر بر اساس الزام قانونی و یا شیوه نامه های داخلی این شرکت چند ملیتی ، کارکنانی که بیش از ۱۰۰۰۰۰ دلار در سال حقوق دریافت می کنند باید ۲۵ درصد مالیات پردازند ، این قاعده تنها شامل کارکنان آمریکای شمالی است و مقایسه حقوق سالانه کارکنان شاغل در ژاپن با مبلغ ۱۰۰۰۰۰ دلار نتیجه نادرستی تولید خواهد کرد.

۲- نوع عملیات مقایسه که می تواند یکی از چند گزینه بزرگتر ، کوچکتر ، برابر (مساوی) ، نابرابر(مخالف) ، بزرگتر مساوی و یا کوچکتر مساوی باشد.

۳- نتیجه حاصل از مقایسه که مهمترین بخش از مقایسه است و در تصمیم گیری و استدلال بکارگرفته میشود ، نتیجه یک مقایسه همواره ماهیت منطقی دارد و تنها می تواند یکی از دو مقدار True و یا False را داشته باشد

عملگرهای مقایسه ای موجود در پایتون بر این پایه اند :

عملگر تساوی یا برابری

این عملگر که با نماد == نشان داده می شود برابری عملوندهای سمت چپ و راست خود را بررسی می کند و چنانچه مقدار عملگر سمت چپ با مقدار عملگر سمت راست برابر باشد مقدار True و در غیر این صورت مقدار False را بر می گرداند

برای مثال فرض کنید وزن دو ورزشکار به نامهای داریوش و آرمان به ترتیب برابر ۷۵ و ۶۹ کیلوگرم باشد ، اگر وزن داریوش و آرمان را به ترتیب در متغیرهای DariushWeight و ArmanWeight ذخیره کنیم ، عملیات مقایسه برابری وزن این دو را در زبان پایتون به شکل زیر می توان نوشت :

DariushWeight = 75

ArmanWeight = 69

Result = (DariushWeight == ArmanWeight)

با توجه به اینکه وزن این دو ورزشکار یکسان نیست پس نتیجه ذخیره شده در متغیر بولی Result مقدار False خواهد بود.

عملگر بزرگتر

این عملگر که با نماد > نشان داده می شود بزرگتر بودن عملوند سمت چپ خود را نسبت به عملوند سمت راست خود بررسی می کند و چنانچه مقدار عملگر سمت چپ بزرگتر از مقدار عملگر سمت راست باشد مقدار True و در غیر این صورت مقدار False را برمی گرداند. در مثال زیر نتیجه ذخیره شده در متغیر Reesult مقدار True خواهد بود

DariushWeight = 75

ArmanWeight = 69

Result = (DariushWeight > ArmanWeight)

عملگر کوچکتر

این عملگر که با نماد < نشان داده می شود کوچکتر بودن عملوند سمت چپ خود را نسبت به عملوند سمت راست خود بررسی می کند و چنانچه مقدار عملگر سمت چپ کوچکتر از مقدار عملگر سمت راست باشد مقدار True و در غیر این صورت مقدار False را بر می گرداند. در مثال زیر نتیجه ذخیره شده در متغیر Reesult مقدار False خواهد بود.

DariushWeight = 75

ArmanWeight = 69

Result = (DariushWeight < ArmanWeight)

عملگر نابرابر

این عملگر که با نامهای نامساوی یا مخالف هم از آن یاد می شود با نماد $=$! نشان داده می شود و نابرابر بودن عملوند سمت چپ خود را نسبت به عملوند سمت راست بررسی می کند و چنانچه مقدار عملگر سمت چپ نابرابر با مقدار عملگر سمت راست باشد مقدار True و در غیر این صورت مقدار False را بر می گرداند. در مثال زیر نتیجه ذخیره شده در متغیر Reesult مقدار True خواهد بود

DariushWeight = 75

ArmanWeight = 69

Result = (DariushWeight != ArmanWeight)

عملگر بزرگتر مساوی

این عملگر، عملیات دو عملگر مقایسه ای بزرگتر ($>$) و برابری ($=$) را ترکیب می کند و با نماد $>=$ (بخوانید بزرگتر مساوی) نشان داده می شود. برای مثال زمانی که گفته می شود کمترین سن اخذ گواهینامه در ایران ۱۸ سال است به این معنی است که افراد ۱۸ سال و بالاتر می توانند گواهی نامه رانندگی درخواست کنند اما افراد زیر ۱۸ سال این حق را نخواهند داشت. این مثال را در پایتون می توان به صورت زیر کد نویسی کرد :

Age = 18

PersonAge = 21

Result = (PersonAge >= Age)

در مثال بالا متغیر Age متغیری است که سن مبنا یا همان کمترین سن لازم برای درخواست گواهی نامه را که در ایران ۱۸ سال تمام است در خود ذخیره می کند و متغیر PersonAge نیز سن افراد متقاضی گواهینامه رانندگی را در برخواهد داشت. با اجرای خط سوم توسط مفسر پایتون ، مقدار True در متغیر Result خواهد نشست. زیرا نتیجه عبارت $PersonAge >= Age$ که مقدار True است با عملگر انتساب ($=$) در متغیر Result نشسته است.

عملگر کوچتر مساوی

این عملگر، عملیات دو عملگر مقایسه ای کوچتر (<) و برابری (==) را ترکیب می کند و با نماد <= (بخوانید کوچتر مساوی) نشان داده می شود. برای نمونه زمانی که گفته می شود بیشینه وامی که بانک مسکن برای خرید خانه می پردازد ۲۰۰ میلیون تومان است به این معنی است که مبلغ وامی که شما می توانید از بانک مسکن بابت خرید خانه درخواست کنید باید کمتر یا مساوی ۲۰۰ میلیون تومان باشد. و شما نمی توانید بیش از ۲۰۰ میلیون تومان وام از این بانک درخواست کنید. از این رو بر اساس نیاز می توانید ۱۵۰ میلیون تومان وام درخواست دهید اما نمی توانید درخواست وام ۲۵۰ میلیون تومانی بدهید. این مثال را در پایتون می توان به صورت زیر کد نویسی کرد :

```
DebtLimit = 200000000
```

```
DebtRequest = 150000000
```

```
Result = (DebtRequest <= DebtLimit)
```

در مثال بالا متغیر DebtLimit متغیری است که سقف وامی که بانک می تواند وام پرداخت کند را نگه می دارد و متغیر DebtRequest نیز وام درخواست شده توسط مشتری را در برخواهد داشت. با اجرای خط سوم توسط مفسر پایتون مقدار True در متغیر Result خواهد نشست. زیرا نتیجه ارزیابی عبارت DebtRequest <= DebtLimit که مقدار True است با عملگر انتساب (=) در متغیر Result نشسته است. اما در مثال زیر مقدار False در متغیر Result می نشیند:

```
DebtLimit = 200000000
```

```
DebtRequest = 250000000
```

```
Result = (DebtRequest <= DebtLimit)
```

متغیرهای نوع منطقی

دانستیم که نتیجه حاصل از مقایسه، مهمترین بخش یک مقایسه است و در تصمیم گیری و استدلال بکار گرفته می شود همچنین گفتیم که نتیجه یک مقایسه همواره ماهیت منطقی دارد و تنها می تواند یکی از دو مقدار درست (True) و یا نادرست (False) را داشته

باشد. از این رو بهترین نوع متغیر که می تواند نتیجه یک مقایسه را در خود ذخیره کند متغیر منطقی^۱ است .

در زبان برنامه نویسی پایتون ، متغیرهای نوع منطقی یا Boolean دسته ای از متغیرها هستند که تنها می توانند دو مقدار منطقی درست (True) و یا نادرست (False) را در خود نگه دارند.

مثال :

IsLast = True

MainStage = False

برای درک بهتر نتیجه حاصل از یک مقایسه و چگونگی ذخیره نتیجه در یک متغیر منطقی ، فرض کنید که میانگین قد دانش آموزان یک مدرسه در واحد سانتی متر در متغیری به نام StudentHeightAverage ذخیره شده است و قد دانش آموزی به نام آرش در متغیری به نام ArashHeight ذخیره می شود برای مقایسه قد آرش با میانگین قد دانش آموزان کلاس از قطعه کد زیر استفاده می کنیم :

StudentHeightAverage = 140

ArashHeight = 150

Result = ArashHeight > StudentHeightAverage

با نگاهی به کد بالا مشخص است که میانگین قد دانش آموزان کلاس ۱۴۰ سانتی متر و قد آرش ۱۵۰ سانتی متر است پس قد آرش بزرگتر از میانگین قد کلاس است بنابراین نتیجه ارزیابی عبارت مقایسه ای ArashHeight > StudentHeightAverage درست (True) است. و در نتیجه چیزی که در متغیر منطقی Result قرار می گیرد مقدار True است.

¹ Boolean

مثال	نام	عملگر
$x == y$	مساوی با	$==$
$x != y$	نابرابر ، مخالف	$!=$
$x > y$	بزرگتر	$>$
$x < y$	کوچکتر	$<$
$x >= y$	بزرگتر مساوی	$>=$
$x <= y$	کوچکتر مساوی	$<=$

جدول ۲-۴ عملگرهای رابطه ای در پایتون

عملگرهای منطقی

فرض کنید مسئول ثبت نام اردوی علمی دانش آموزان یک مدرسه دخترانه هستید و مدیرمدرسه از شما می خواهد که تنها از دانش آموزانی برای شرکت در اردو ثبت نام کنید که در کارنامه امتحانات نوبت اول دارای معدل بالای ۱۶ باشند و نمره درس علوم آنها کمتر از ۱۷ نباشد. همانگونه که می بینید مدیر این مدرسه دو شرط لازم برای ثبت نام در اردوی علمی را با حرف ربط عطفی ((و)) به هم پیوند داده است و این یعنی: تنها دانش آموزانی می توانند در این اردو شرکت کنند که هر دو شرط معدل بالای ۱۶ و نمره درس علوم بالاتر از ۱۷ یا بالاتر را باهم داشته باشند. و اگر دانش آموزی یکی از این دو شرط را نداشته باشد نمی تواند در اردوی علمی ثبت نام کند.

حال فرض کنید دستور مدیر مدرسه به این شکل تغییر کند که دانش آموزانی می توانند در اردوی علمی مدرسه شرکت کنند که در کارنامه امتحانات نوبت اول دارای معدل بالای ۱۶ باشند یا نمره درس علوم آنها کمتر از ۱۷ نباشد. همانگونه که می بینید تنها چیزی که در دستور جدید تغییر کرده است حرف ربطی است که دو شرط مدیر مدرسه را به هم پیوند می دهد. در این دستور مدیر دو

شرط مورد نظر خود را با حرف ربط فصلی (یا) به هم پیوند داده است یعنی هر دانش آموزی که دارای معدل بالای ۱۶ است می تواند در این اردو شرکت کند حتی اگر نمره درس علوم وی کمتر از ۱۷ باشد و نیز هر دانش آموزی که نمره درس علوم ۱۷ یا بالاتر داشته باشد می تواند در این اردو شرکت کند حتی اگر معدل وی کمتر از ۱۶ باشد.

همانند حروف ربط عطفی (و) و فصلی (یا) که در منطق و ریاضیات دو یا چند گزاره را با هم ترکیب می کنند و بار معنایی و نتیجه ارزیابی گزاره ترکیبی را تغییر می دهند در زبان های برنامه نویسی هم دسته ای از عملگرها به نام عملگرهای منطقی وجود دارند که وظیفه پیوند دادن دو یا چند عبارت منطقی ساده و ایجاد یک عبارت منطقی ترکیبی را برعهده دارند. عملگرهای منطقی نتیجه ارزیابی کل عبارت ترکیب شده را تغییر می دهند. یک عبارت منطقی گزاره ای است که نتیجه ارزیابی آن تنها می تواند یکی از دو مقدار منطقی True یا False باشد. در ادامه به عملگرهای منطقی موجود در زبان برنامه نویسی پایتون می پردازیم :

عملگر منطقی and

در زبان برنامه نویسی پایتون عملگر عطف منطقی یا همان (و) که با نماد and نمایش داده می شود یک عملگر منطقی دو عملوندی است و نتیجه ارزیابی عبارت ترکیبی حاصل از آن تنها در صورتی True است که هر دو عملوند آن دارای ارزش True باشند در غیر این صورت نتیجه ارزیابی False خواهد بود برای درک بهتر این مطلب فرض کنید عملوند های سمت چپ و راست این عملگر را به ترتیب با p و q نمایش می دهیم برای نمایش ارزش عبارت ترکیبی حاصل از عملگر منطقی and بر اساس ارزش عملوندهای آن می توان از جدول زیر استفاده کرد :

P	q	P and q
درست	درست	درست
درست	نادرست	نادرست
نادرست	نادرست	نادرست
نادرست	درست	نادرست

جدول ۳-۴ جدول نمایش ارزیابی عملگر and

بنابراین ارزش عبارت ترکیبی حاصل از عملگر منطقی and تنها در صورتی True است که هر دو عملوند آن دارای ارزش True باشند در غیر این صورت دارای ارزش False است.

مثال :

کشور ایران در قاره آسیا قرار دارد و پایتخت ژاپن شهر لندن است

اگر گزاره کشور ایران در قاره آسیا قرار دارد را با p و گزاره پایتخت کشور ژاپن شهر لندن است را با q نمایش دهیم می توان گفت که گزاره p دارای ارزش درست است زیرا کشور ایران در قاره آسیا قرار دارد اما گزاره q دارای ارزش نادرست است چرا که لندن پایتخت کشور ژاپن نیست بنابراین بر اساس جدول بالا ارزش عبارت منطقی p and q نادرست است.

حالت کلی	P	q	P and q
مثال	کشور ایران در قاره آسیا قرار دارد	پایتخت کشور ژاپن شهر لندن است	کشور ایران در قاره آسیا قرار دارد And پایتخت کشور ژاپن شهر لندن است
ارزش	True	False	False

مثال : فرض کنید مینا دانش آموزی است که دارای معدل ۱۸ و نمره علوم ۱۹ است با فرض اینکه تنها دانش آموزانی می توانند در اردو شرکت کنند که دارای معدل بالاتر از ۱۶ باشند و نمره درس علوم آنها کمتر از ۱۷ نباشد آیا وی می تواند در اردوی علمی شرکت کند؟

حل :

StudentAverage = 18

StudentGrade = 19

Result = (StudentAverage>16) and (StudentGrade> = 17)

Print(Result)

StudentAverage متغیری است که معدل دانش آموز را در خود نگه دارد و StudentGrade هم متغیری است که نمره علوم دانش آموز را در خود ذخیره می کند بنا براین عبارت سمت چپ عملگر and یعنی StudentAverage>16 دارای ارزش True است زیرا معدل مینا ۱۸ است و عدد ۱۸ بزرگتر از ۱۶ است. همچنین عبارت سمت راست عملگر and یعنی StudentGrade> = 17 نیز دارای ارزش True است چراکه نمره درس علوم مینا ۱۹ است و عدد ۱۹ بزرگتر از عدد ۱۷ است بنابراین بر اساس جدول بالا حالت اول رخ داده و p و q یعنی عملوندهای سمت چپ و راست عملگر منطقی and هر دو True هستند از این رو نتیجه عملگر and در مثال بالا True است که از طریق عملگر انتساب = در متغیر Result می نشیند و در نتیجه خروجی چاپ شده توسط دستور Print(Result) هم True است. و این به معنای این است که مینا هردو شرط مورد انتظار برای شرکت در اردوی علمی مدرسه را داراست و می تواند در اردو حضور یابد.

حالت کلی	P	q	P and q
مثال	StudentAverage>16	StudentGrade> = 17	(StudentAverage>16) and (StudentGrade> = 17)
ارزش	True	True	True

مثال :

$x = 5$

`print(x > 3 and x < 10)`

در مثال بالا دستور `print` مقدار `True` را چاپ خواهد کرد زیرا مقدار متغیر x یعنی عدد ۵ بزرگتر از عدد ۳ و کوچکتر از عدد ۱۰ است بنابراین عملوندهای سمت چپ و راست عملگر منطقی `and` هر دو دارای ارزش درست هستند پس خروجی این عملگر `True` می باشد.

حالت کلی	P	q	P and q
مثال	$x > 3$	$x < 10$	$X > 3 \text{ and } x < 10$
ارزش	True	True	True

عملگر منطقی OR

در زبان برنامه نویسی پایتون عملگر فصل منطقی یا همان (یا) که با نماد `or` نمایش داده می شود یک عملگر منطقی دو عملوندی است و نتیجه عبارت ترکیبی حاصل از آن تنها در صورتی `False` است که هر دو عملوند آن دارای ارزش `False` باشند در غیر این صورت نتیجه ارزیابی عبارت ترکیبی حاصل از آن دارای ارزش `True` خواهد بود برای درک بهتر این مطلب فرض کنید عملوندهای سمت چپ و راست این عملگر را به ترتیب با p و q نمایش دهیم برای نمایش ارزش عبارت ترکیبی حاصل از عملگر `or` بر اساس ارزش عملوندهای آن می توان از جدول زیر استفاده کرد :

P	q	P or q
درست	درست	درست
درست	نادرست	درست
نادرست	نادرست	نادرست
نادرست	درست	درست

جدول ۴-۴ جدول نمایش ارزیابی عملگر or

بنابراین ارزش عبارت ترکیبی حاصل از عملگر منطقی or تنها در صورتی False است که هر دو عملوند آن دارای ارزش False باشند، در غیراین صورت دارای ارزش خروجی True است.

مثال :

عدد ۱۶ زوج است یا عدد ۱۷ بزرگتر از عدد ۳۶ است

اگر گزاره عدد ۱۶ زوج است را با p و گزاره عدد ۱۷ بزرگتر از عدد ۳۶ است را با q نمایش دهیم می توان گفت که گزاره p دارای ارزش درست است زیرا ۱۶ یک عدد زوج است چراکه باقی مانده تقسیم آن بر ۲ صفر است اما گزاره q دارای ارزش نادرست است زیرا عدد ۱۷ بزرگتر از عدد ۳۶ نیست بنابراین بر اساس جدول بالا ارزش عبارت منطقی p or q درست است.

حالت کلی	P	q	P or q
مثال	عدد ۱۶ زوج است	عدد ۱۷ بزرگتر از ۳۶ است	عدد ۱۶ زوج است or عدد ۱۷ بزرگتر از عدد ۳۶ است
ارزش	True	False	True

مثال : فرض کنید رشته دانش آموزی است که دارای معدل ۱۵ و نمره علوم ۱۴ است با فرض اینکه تنها دانش آموزانی می توانند در اردوی علمی مدرسه شرکت کنند که دارای معدل بالاتر از ۱۶ باشند یا نمره درس علوم آنها کمتر از ۱۷ نباشد آیا وی می تواند در اردوی علمی شرکت کند؟

حل :

StudentAverage = 15

StudentAverage = 14

Result = (StudentAverage>16) or (StudentGrade> = 17)

Print(Result)

StudentAverage متغیری است که معدل دانش آموز را در خود نگه دارد و StudentGrade هم متغیری است که نمره علوم دانش آموز را در خود ذخیره می کند بنا براین عبارت سمت چپ عملگر or یعنی StudentAverage>16 دارای ارزش False است زیرا معدل رشته ۱۵ است و عدد ۱۵ کوچکتر از عدد ۱۶ است. همچنین عبارت سمت راست عملگر or یعنی StudentGrade> = 17 نیز دارای ارزش False است چراکه نمره درس علوم رشته ۱۴ است و عدد ۱۴ کوچکتر از عدد ۱۷ است بنابراین بر اساس جدول درستی عملگر or در بالا حالت سوم رخ داده و p و q یعنی عملوندهای سمت چپ و راست عملگر منطقی or هر دو False هستند از این رو خروجی عملگر or در مثال بالا False است که از طریق عملگر انتساب = در متغیر Result می نشیند و در نتیجه خروجی چاپ شده توسط دستور Print(Result) هم مقدار False است. و این به معنای این است که رشته هردو شرط مورد انتظار برای شرکت در اردوی علمی مدرسه را ندارد و بنابراین نمی تواند در اردو حضور یابد.

حالت کلی	P	q	P or q
مثال	StudentAverage>16	StudentGrade>= 17	(StudentAverage>16) or (StudentGrade>= 17)
ارزش	False	False	False

مثال :

x = 8

print(x > 3 or x < 5)

در مثال بالا دستور print مقدار True را چاپ خواهد کرد زیرا مقدار متغیر x یعنی عدد ۸ بزرگتر از عدد ۳ است بنابراین عملوند سمت چپ عملگر or دارای ارزش درست است اما چون عدد ۸ کوچکتر از عدد ۵ نیست پس عملوند سمت راست عملگر or دارای ارزش نادرست است بنابراین حالت دوم جدول درستی عملگر or رخ داده است پس خروجی این عملگر True می باشد.

حالت کلی	P	q	P or q
مثال	x>3	x<5	X > 3 and x < 5
ارزش	True	False	True

عملگر منطقی not

یکی دیگر از عملگرهای منطقی در زبان برنامه نویسی پایتون ، عملگر منطقی not است ، این عملگر یک عملگر یکانی است یعنی برخلاف عملگرهای منطقی and و or که دارای دو عملوند هستند تنها دارای یک عملوند است. وظیفه اصلی عملگر not عکس کردن ارزش درستی عملوند خود است یعنی چنانچه عملوند آن دارای ارزش True باشد خروجی این عملگر عکس ارزش عملوند آن یعنی False است و اگر ارزش عملوند آن False باشد خروجی این عملگر True است. جدول ارزش درستی عملگر منطقی not به شکل زیر است :

P	not p
درست	نادرست
نادرست	درست

جدول ۴-۵ جدول نمایش ارزیابی عملگر not

مثال : خروجی دستور print در برنامه زیر True است

```
x = 5
```

```
y = not (x>12)
```

```
print(y)
```

عدد ۵ کوچکتر از عدد ۱۲ است پس نتیجه عبارت مقایسه ای $x > 12$ نادرست (False) است بنابراین خروجی عملگر منطقی not عکس آن یعنی True خواهد بود که توسط عملگر انتساب = در متغیر y قرار خواهد گرفت و در خط آخر توسط دستور print(y) چاپ می شود.

P	q	P or q	P and q	not p	not q
درست	درست	درست	درست	نادرست	نادرست
درست	نادرست	درست	نادرست	نادرست	درست
نادرست	نادرست	نادرست	نادرست	درست	درست
نادرست	درست	درست	نادرست	درست	نادرست

جدول ۴-۶ جدول نمایش ارزیابی عملگرهای منطقی

عملگرهای انتساب ترکیبی

با نگاهی به عبارت $X = X + 1$ می توان دریافت که متغیر X در دو طرف عملگر انتساب (=) تکرار شده است در این گونه گزاره ها که عملیات خاصی (برای مثال عمل جمع) در سمت راست عملگر انتساب بر روی یک متغیر انجام می شود و نتیجه حاصل در خود آن متغیر ریخته می شود می توان از عملگرهای انتساب ترکیبی استفاده کرد.

مثال: عبارت $y = y * 5$ را می توان به شکل $y * = 5$ بازنویسی کرد.

مثال : عبارت $a = a + b + c$ را می توان به شکل $a + = (b + c)$ نوشت.

جدول برخی از مهمترین عملگرهای انتساب ترکیبی		
عملگر انتساب ترکیبی	مثال ترکیبی	مثال غیر ترکیبی
$+ =$	$x + = 3$	$x = x + 3$
$- =$	$x - = 3$	$x = x - 3$
$* =$	$x * = 3$	$x = x * 3$
$/ =$	$x / = 3$	$x = x / 3$
$\% =$	$x \% = 3$	$x = x \% 3$
$// =$	$x // = 3$	$x = x // 3$
$** =$	$x ** = 3$	$x = x ** 3$

فصل پنجم

تصمیم گیری و هدایت جریان برنامه

به عنوان یک برنامه نویس تنها زمانی به ارزش عملگرهای منطقی و مقایسه ای پی خواهید برد که با مفهوم هدایت جریان برنامه یا همان تصمیم گیری در زبانهای برنامه نویسی آشنا شوید. چیزی که موضوع اصلی این فصل است.

تصمیم گیری مفهومی آشنا و پرکاربرد در زندگی شخصی و کاری هر فرد است. و همه ما در لحظه لحظه زندگی خود ناچار به تصمیم گیری و اقدام بر اساس آن هستیم. برای نمونه مدیر کسب و کاری که با توجه به شرایط بازار، دستور پایان دادن به تولید یک محصول قدیمی را می دهد و یا بانوی خانه داری که نوع غذای ناهار خانواده را انتخاب می کند تا کارمندی که مسیر رانندگی تا اداره را با توجه شرایطی چون زمان باقی مانده به ساعت شروع کار، خلوتی خیابانها و ... انتخاب می کند، همگی در حال تصمیم گیری هستند. هرچند که ممکن است شیوه و سطح تصمیم گیری هر یک از ما با دیگری متفاوت باشد اما بسیاری از تصمیماتی که در دنیای شخصی و یا حتی سازمانی گرفته می شوند را می توان بر مبنای اصل انتخاب بر اساس شرایط تفسیر کرد. به بیان ساده تر فرد تصمیم گیرنده شرایطی را بررسی می کند و در صورت برقراری آن شرایط رفتاری را انتخاب می کند. برای مثال بخش کوچکی از رفتار یک راننده به هنگام رانندگی، به شرایطی به نام رنگ چراغ راهنمایی بستگی دارد، اگر چراغ راهنمایی قرمز باشد راننده رفتار ((بازداشتن خودرو از حرکت)) را انتخاب می کند و اگر چراغ راهنمایی سبز باشد راننده رفتار ((به حرکت خود ادامه دهید)) را انتخاب خواهد کرد. به بیانی دیگر راننده بر اساس شرایطی به نام رنگ چراغ راهنمایی رفتار خود را انتخاب می کند. یعنی تصمیم می گیرد که چه کاری انجام دهد. الگوی تصمیم گیری یک راننده با شرایط رنگ چراغ راهنمایی را می توان با دو قاعده اگر ... آنگاه ... زیر نشان داد:

اگر چراغ راهنمایی سبز است **آنگاه** به حرکت خود ادامه دهید

اگر چراغ راهنمایی قرمز است **آنگاه** خودرو را از حرکت باز بدارید.

الگوی بسیاری از تصمیم گیری های ما در زندگی بر اساس قاعده اگر ... آنگاه شکل می گیرد یعنی با استفاده از بخش اگر ، شرط یا شرایطی را بررسی می کنیم و چنانچه این شرط یا شرایط برقرار بودند ، یک یا چند اقدام ممکن را انجام خواهیم داد. برای درک بهتر این الگو مثالهای زیر می توانند سودمند باشند:

مثال ۱: تصمیم گیری یک پزشک برای تجویز دارو برای تب

اگر دمای بدن بیمار بالای ۳۸ درجه سانتی گراد است و بیمار بزرگسال است **آنگاه** به بیمار قرص آسپرین بدهید.

مثال ۲ : تصمیم گیری یک باغبان برای سم پاشی درختان باغ

اگر اواخر زمستان یا اوایل بهار است و هوا بارانی نیست **آنگاه** درختان را سم پاشی کنید.

مثال ۳: تصمیم گیری یک فعال بازار

اگر تورم در حال افزایش است **آنگاه** برای حفظ ارزش پول خود طلا یا ملک بخرید.

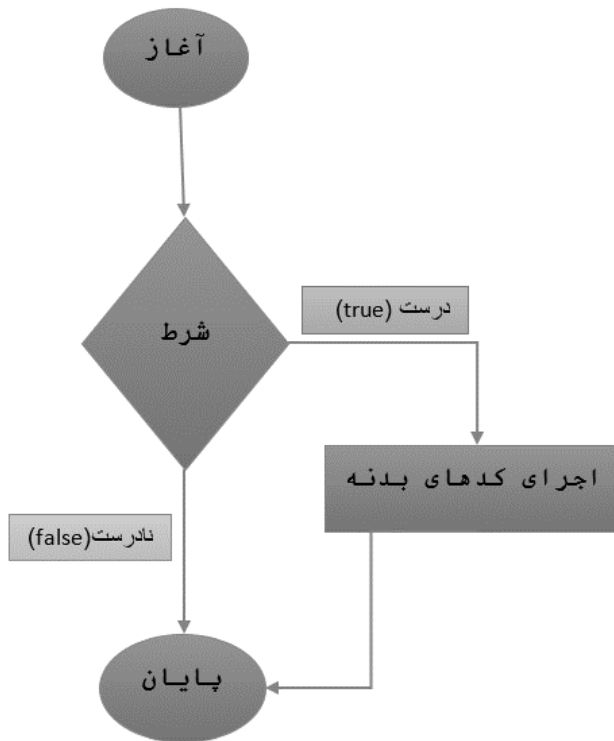
مثال ۴ : تصمیم گیری در اداره مالی یک دانشگاه

اگر دانشجو جزو دانشجویان ممتاز در سال تحصیلی قبل است یا دارای حکم قهرمانی کشوری در یکی از رشته های ورزشی است **آنگاه** ده درصد تخفیف به شهریه این ترم دانشجو داده شود.

همانند دنیای انسانها ، در دنیای برنامه نویسی پایتون هم ، تصمیم گیری و هدایت جریان برنامه با استفاده از اصل انتخاب بر اساس شرایط انجام می شود. ابزار پایتون برای پیاده سازی این اصل ، دو دستور **if** و **match** هستند که در ادامه با آنها بیشتر آشنا خواهید شد.

دستور if

ساده ترین روش برای تصمیم گیری و هدایت جریان برنامه در زبان پایتون بکارگیری دستور **if** است با استفاده از این دستور می توان مفسر پایتون را وادار کرد تا به هنگام اجرای برنامه یک و یا بیش از یک شرط را بررسی کند و در صورت درست بودن شرط یا شرط های بررسی شده ، یک یا چند دستور نوشته شده درون بدنه دستور **if** را اجرا کند.



تصویر ۱-۵ روند نمای دستور if

هر دستور ساده if از چهار بخش اصلی تشکیل شده است :

۱- رهنمون if

۲- بخش شرط که می تواند یک و یا بیش از یک گزاره منطقی ، مقایسه ای و یا آمیزه ای از هر دو نوع را در برداشته باشد. به زبان ساده می توان گفت که در بخش شرط یک دستور if تنها می توان از عملگرهای منطقی و یا مقایسه ای استفاده کرد.

۳- بخش آنگاه که با نویسه دو نقطه بیانی (:) نشان داده می شود و پس از بخش شرط قرار می گیرد.

۴- بدنه if که می تواند یک و یا بیش از یک دستور و عبارت زبان پایتون را در بر داشته باشد. دستورات موجود در بدنه باید با استفاده از یک یا چند فاصله نسبت به if

تورفتگی داشته باشند. مقدار تورفتگی را برنامه نویس به دلخواه انتخاب می کند اما باید تا پایان قطعه کد بدنه ثابت باقی بماند. بیشتر برنامه نویسان برای ایجاد تورفتگی از چهار فاصله خالی استفاده می کنند. پس به یاد داشته باشید که بدنه `if` با تورفتگی نسبت به خود دستور آغاز می شود. و با اولین دستور بدون تورفتگی پایان می یابد.

شکل کلی دستور `if`:

: بخش شرط `if`

بدنه

مثال :

`a = 20`

`b = 15`

`if a > b:`

`print ("a greater than b")`

لازم به یادآوری است که در بیشتر زبان های برنامه نویسی یک قطعه کد مستقل (برای مثال بدنه دستور `if`) را با استفاده از نشانه کمانک باز و بسته `{}` از دیگر بخش های کد جدا می کنند یعنی شروع قطعه کد با کمانک باز `{` و پایان آن با کمانک بسته `}` مشخص می گردد. برای مثال بدنه دستور `if` در زبان برنامه نویسی `C++` به شکل زیر است :

(شرط یا شرطها) `if`

`{`

دستورات درون بدنه

`}`

در زبان پایتون یک قطعه کد مستقل^۱ با ایجاد تورفتگی یکسان در تمامی خط کدهای موجود در قطعه کد مورد نظر صورت می گیرد برای مثال بدنه دستوراتی چون `if`، `while`، `else`، `for` و بسیاری دیگر از دستورات یک قطعه کد مستقل است از این رو بدنه آنها با تورفتگی نسبت به خود دستور آغاز می شود و با اولین دستور بدون تورفتگی پایان می یابد. مقدار تورفتگی را برنامه نویس به دلخواه انتخاب می کند اما باید تا پایان قطعه کد ثابت باقی بماند. بیشتر برنامه نویسان برای ایجاد تورفتگی از چهار فاصله خالی استفاده می کنند.

بخش `else` در دستور `if`

در برخی از کاربردها نیاز است تا در صورت برقرار نبودن بخش شرط دستور `if`، کار دیگری انجام شود، منظور از برقرار نبودن بخش شرط دستور `if` این است که نتیجه ارزیابی عبارت منطقی یا مقایسه ای موجود در بخش شرط `False` باشد، برای مثال اگر عددی که کاربر وارد کرده است زوج باشد باید دستور یا دستوراتی اجرا گردد وگرنه دستور و یا دستورات دیگری اجرا خواهند شد. در این حالت شکل کلی دستور `if` همانند الگوی زیر خواهد بود :

اگر شرط/ شرطها **آنگاه** دستور/ دستورات **وگرنه** دستور/ دستورات

که به زبان پایتون همانند نمونه زیر نوشته می شود :

شرطها `if` :

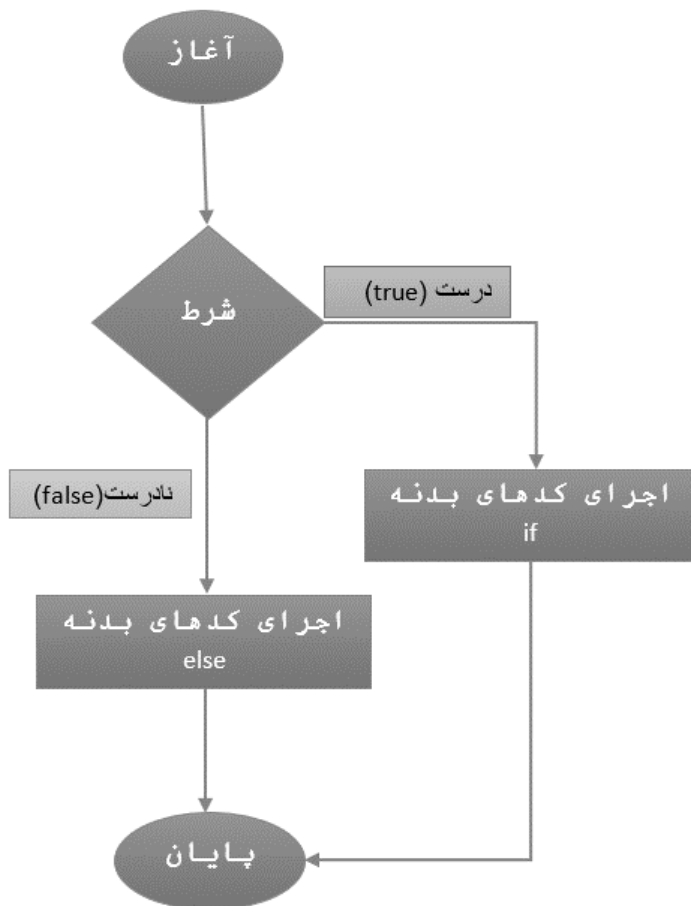
دستورات

`else`

دستورات

لازم به گفتن است که رعایت اصل تورفتگی قطعه کد بدنه در بخش `else` دستور `if` الزامی است. نمودار جریان داده دستور `if else` به شکل زیر است :

¹ Code Block



تصویر ۲-۵ روند نمای دستور if else

مثال : برنامه ای بنویسید که زوج یا فرد بودن عددی را بررسی کرده و چنانچه عدد زوج باشد مربع آنرا چاپ کند وگرنه خود عدد را در خروجی نمایش دهد.

```
num = 80
if num % 2 == 0 :
    Print(num*num)
else
    Print(num)
```

مثال : برنامه ای بنویسید که قدر مطلق یک عدد را در خروجی چاپ کند

حل : می دانیم که از نظر ریاضی قدر مطلق یک عدد حقیقی x که با نماد $|x|$ نمایش داده می شود یک عدد همواره مثبت و یا صفر است بنا بر این چنانچه x یک عدد حقیقی مثبت و یا صفر باشد آنگاه $|x| = x$ و چنانچه x یک عدد حقیقی منفی باشد آنگاه $|x| = -x$ برای مثال :

$$|2| = 2$$

$$|-5| = -(-5) = +5$$

$$|0| = 0$$

حال با داشتن این آگاهی در باره قدر مطلق یک عدد می توان برنامه زیر را برای محاسبه قدر مطلق نوشت :

```
number = 55
```

```
if number >= 0 :
```

```
    print (number)
```

```
else
```

```
    print (- number)
```

بخش elif در دستور if

گاهی نرم افزار ناچار است تا بر اساس حالت های مختلف یک مسئله تصمیم گیری کند و بسته به اینکه ورودی های مسئله با شرایط تعریف شده در کدام یک از حالت های تعریف شده سازگار است کار جداگانه ای را به سرانجام رساند. برای مثال فرض کنید شرکت برق برای محاسبه بهای برق مصرفی مشترکین خانگی چهار حالت مختلف را براساس میزان مصرف مشترک در بازه زمانی یک ماهه تعریف و برای هر یک تعرفه ویژه ای همانند زیر در نظر گرفته است:

۱- تعرفه مصرف تا ۱۰۰ کیلووات ساعت (kwh) برابر ۵۲۴ ریال برای هر kwh است.

۲- تعرفه مصرف از ۱۰۱ تا ۲۰۰ کیلو وات ساعت برابر ۶۱۱ ریال برای هر kwh است.

۳- تعرفه مصرف از ۲۰۱ تا ۳۰۰ کیلو وات ساعت برابر ۱۳۱۰ ریال برای هر kwh است.

۴- تعرفه مصرف بیش از ۳۰۰ کیلووات ساعت برابر ۱۵۵۰ ریال برای هر kwh است.

ساده ترین راه برای نوشتن برنامه ای که بتواند بهای برق مصرفی یک مشترک خانگی را محاسبه و چاپ کند استفاده از چهار دستور ساده if است اما پایتون راه حل بهتری برای حل چنین مسائلی دارد و آن استفاده از بخش elif در دستور if است که در برنامه زیر بکار رفته است. elif ساده شده عبارت else if است که به صورت در غیر این صورت اگر خوانده می شود.

CustomerKWH = 280

TotalCost = 0

if CustomerKWH <= 100 :

TotalCost = CustomerKWH * 524

elif CustomerKWH >101 and CustomerKWH<= 200 :

TotalCost = CustomerKWH * 611

elif CustomerKWH >201 and CustomerKWH<= 300 :

TotalCost = CustomerKWH * 1310

elif CustomerKWH >300 :

TotalCost = CustomerKWH * 1550

print(TotalCost)

مثال : شاخص توده بدنی (BMI) یک معیار مهم در پزشکی است و برای اندازه گیری سلامت وزن افراد نسبت به قد آنان بکار می رود. به کمک این شاخص می توان گفت

که فرد دارای اضافه وزن و یا کمبود وزن است . برای محاسبه BMI کافیست تا وزن فرد را بر مربع قد وی تقسیم کنید یکای اندازه گیری وزن باید کیلوگرم (Kg) و یکای اندازه گیری قد باید متر (m) باشد، بسته به اینکه BMI محاسبه شده چه مقداری داشته باشد یک از چهار حالت زیر رخ خواهد داد :

۱- BMI کمتر از ۱۸,۵ به معنی کمبود وزن است

۲- BMI بین ۱۸,۵ تا ۲۴,۵ به معنای سلامت وزن است

۳- BMI بین ۲۵ تا ۲۹,۹ به معنای اضافه وزن است

۴- BMI بیشتر از ۳۰ به معنای چاقی است.

حال بر اساس اطلاعات بالا برنامه ای بنویسید که با داشتن سن دانش آموزان در یکای کیلوگرم و قد آنان در یکای متر بتواند BMI دانش آموز را محاسبه و وضعیت سلامت وزن وی را در خروجی چاپ کند:

```
StudentHeight = 1.5
StudentWeight = 45
BMI = StudentWeight/ StudentHeight**2
if BMI<18.5 :
    print("فرد دارای کمبود وزن است")
elif BMI >= 18.5 and BMI <= 24.5 :
    print("فرد سلامت وزن دارد")
elif BMI >= 25 and BMI <= 30 :
    print("فرد دارای اضافه وزن است")
elif BMI>30
    print("فرد چاق است")
```

مثال : المپیک ۲۰۲۰ توکیو در زمانی برگزار گردید که جهان با بحران همه گیری کرونا دست به گریبان بود ، در این رویداد جهانی بیش از ده هزار ورزشکار از کشورهای مختلف جهان شرکت کردند از این رو یکی از مهمترین چالش های فراروی دست

اندرکاران برگزاری این مسابقات این بود که ورزشکاران در چه مکانی و در چه شرایطی باید اسکان داده شوند تا فاصله گذاری اجتماعی بیشینه و انتقال ویروس در بین ورزشکاران کمینه گردد. سرانجام ژاپنی ها تصمیم گرفتند تا با ایجاد یک دهکده بزرگ به مساحت بیش از ۴۴ هکتار و دارای ۲۱ ساختمان مسکونی مجهز به ۳۸۰۰ واحد اقامتی بر این چالش چیره شوند فرض کنید انتقال ورزشکاران به هریک از این ساختمان ها بر اساس کشور ورزشکار انجام می شود برنامه ای بنویسید که بر اساس حالت های ذیل ساختمان محل اسکان هر ورزشکار را در خروجی چاپ کند. (برای راحتی کار تنها چهار حالت در نظر گرفته شده است در حالی که برنامه واقعی به ۲۱ حالت نیاز خواهد داشت)

۱- ورزشکاران کشورهای ایران ، یونان و ارمنستان در ساختمان شماره ۱

۲- ورزشکاران آمریکا ، ایتالیا و آلمان در ساختمان شماره ۲

۳- ورزشکاران هند ، مصر و چین در ساختمان شماره ۳

۴- ورزشکاران دیگر کشورها در ساختمانی به انتخاب مدیراجرایی دهکده المپیک اسکان داده خواهند شد.

```
country = "iran"
if country == "iran" or country == "greece" or country == "armenia" :
    print("اسکان در ساختمان شماره ۱")
elif country == "united state" or country == "italy" or country == "germany" :
    print("اسکان در ساختمان شماره ۲")
elif country == "india" or country == "egypt" or country == "china" :
    print("اسکان در ساختمان شماره ۳")
else
    print("بر اساس انتخاب مدیر اجرایی")
```

همانگونه که در مثال های بالا دیده می شود ، حالت کلی استفاده از بخش **elif** در ساختار دستور **if** همانند الگوی زیر است :

if شرط :

دستورات

elif شرط :

دستورات

elif شرط :

دستورات

else:

دستورات

در این الگو، مفسر پایتون شرط ها را از بالا به پایین بررسی می کند و با رسیدن به نخستین شرطی که نتیجه ارزیابی آن **True** است دستورات موجود در بدنه آن شرط را اجرا می کند و با خروج از ساختار **if** دیگر شرط ها را نادیده گرفته و از ارزیابی آنها خودداری خواهد کرد. اگر نتیجه ارزیابی هیچ یک از شرط ها **True** نباشد، دستور موجود در بدنه **else** پایانی اجرا خواهد شد و اگر **else** پایانی در الگوی **elif** وجود نداشت آنگاه مفسر از ساختار **if** خارج شده و با اجرای اولین دستور پس از آن ، اجرای برنامه اصلی را پی خواهد گرفت.

If های تودرتو

گاهی نیاز است تا در صورت برقرار بودن یا برقرار نبودن یک شرط اصلی در دستور **if** ، شرط یا شرط های فرعی دیگری در داخل بدنه **if** بررسی گردند ، به زبان ساده تر ، می توان یک یا چند دستور **if** را درون بدنه دستور **if** دیگری نوشت که به این حالت **if** های تودرتو گفته می شود. برای مثال فرض کنید که یک سازمان دولتی هزینه ماموریت کارکنان خود را بر اساس شیوه نامه زیر پرداخت می کند :

۱- اگر کارمند با خودروی دولتی دارای راننده به ماموریت اعزام گردد بر اساس فاصله مبدا تا مقصد هزینه ماموریت روزانه به صورت زیر پرداخت گردد :

کمتر از ۱۰ کیلومتر ۰ ریال

از ۱۰ تا ۵۰ کیلومتر ۷۵۰۰۰۰ ریال

بیش از ۵۰ کیلومتر ۱۱۵۰۰۰۰ ریال

۲- اگر کارمند با خودروی دولتی و بدون راننده به ماموریت اعزام گردد بر اساس فاصله مبدا تا مقصد ، هزینه ماموریت روزانه به صورت زیر پرداخت گردد :

کمتر از ۱۰ کیلومتر ۲۵۰۰۰۰ ریال

از ۱۰ تا ۵۰ کیلومتر ۱۲۵۰۰۰۰ ریال

بیش از ۵۰ کیلومتر ۱۷۵۰۰۰۰ ریال

همانگونه که از متن این شیوه نامه فرضی بدست می آید ، شرط اصلی این است که آیا کارمند با راننده و یا بدون راننده به ماموریت اعزام می گردد و شرط فرعی هم مسافت مبدا تا مقصد ماموریت است. از این رو با استفاده از if های تودرتو می توان برنامه ای نوشت که هزینه ماموریت روزانه کارکنان این سازمان را به صورت زیر محاسبه و چاپ کند :

کارمند به همراه راننده به ماموریت اعزام شده است یا بدون راننده # HaveDriver = False

فاصله مبدا تا مقصد به کیلومتر # Distance = 75

هزینه ماموریت روزانه # Cost = 0

if HaveDriver == True:

if Distance < 10:

Cost = 0

elif Distance >= 10 and Distance <= 50:

Cost = 750000

else:

Cost = 1150000

```

else:
    if Distance < 10:
        Cost = 250000
    elif Distance >= 10 and Distance <= 50:
        Cost = 1250000
    else :
        Cost = 1750000
print(Cost)

```

دستور match

دستور match مقدار یک متغیر را با چندین مقدار دیگر که در هر شاخه دستور (هر شاخه case نامیده می شود) آورده می شوند مقایسه می کند اگر نتیجه منطقی بدست آمده از مقایسه True باشد کد نوشته شده در آن شاخه اجرا می شود و اجرای دستور پایان می یابد، چنانچه نتیجه مقایسه در هیچ یک از شاخه ها True نباشد کد موجود در شاخه ویژه ای که به صورت : _ case نوشته می شود اجرا خواهد شد. در این شاخه نویسه خط زیر (_) به معنای سایر مقدارهاست و نتیجه مقایسه منطقی آن با هر مقداری همواره True است. شیوه نگارش و الگوی کلی بکارگیری این دستور همانند زیر است :

```

match variable:
    case value1:
        code
    case value2:
        code
    case value3:
        code
    case _:
        code

```

مثال : بررسی خطاهای معمول در درخواستهای HTTP

```
status = 404
match status:
    case 400:
        print("Bad Request")
    case 401:
        print("Unauthorized")
    case 404:
        print("Not Found")
    case 405:
        print("Method Not Allowed")
    case _:
        print("Somthing's Wrong")
```

مثال : کد بالا رادر قالب یک تابع به نام `http_status_error` می نویسیم با مفهوم تابع در درسهای بعد آشنا خواهید شد :

```
def http_status_error(status):
    match status:
        case 400:
            return "Bad Request"
        case 401:
            return "Unauthorized"
        case 404:
            return "Not Found"
        case 405:
            return "Method Not Allowed"
        case _:
            return "Somthing's Wrong"

print(http_status_error(405))
```

در هر شاخه می توان مقدارهای مختلفی را با استفاده از عملگر | که همان عملگر منطقی or است ترکیب کرد :

مثال :

```
case 401 | 403 | 404:
    print("Not Allowed")
```

مثال :

```
def check_point(point):
    match point:
        case (0, 0):
            print("Origin")
        case (0, y):
            print(f"y={y}")
        case (x, 0):
            print(f"x={x}")
        case (x, y):
            print(f"x={x} y={y}")
        case _:
            raise ValueError("Not A Pointer")

p1 = (30,60)
p2 = (0,0)
p3 = (0,25)
p4 = (37,0)
check_point(p1)
check_point(p2)
check_point(p3)
check_point(p4)
```

خروجی کد بالا همانند زیر است :

```
x=30 y=60
Origin
y=25
x=37
```

فصل ششم

آشنایی با دستور range، عملگرهای عضویت و تبدیل نوع

یک دنباله عددی، رشته ای از اعداد است که در آن هر عدد با افزودن یک عدد ثابت مخالف با صفر به عدد پیش از خود بدست می آید. برای نمونه رشته اعداد 3, 6, 9, 12, 15 یک دنباله عددی است که با عدد ۳ آغاز شده و با گامهای افزایشی ۳ واحدی رشد کرده است. یعنی هر عدد با افزودن عدد ۳ به عدد قبلی بدست آمده است. آخرین عدد این دنباله یعنی عددی که دنباله در آن پایان یافته است. عدد ۱۵ است. در زبان برنامه نویسی پایتون، برای ساخت دنباله ای از اعداد دلخواه از دستور `range()` استفاده می شود. واژه درست تر برای اشاره به این دستور، تابع `range()` است که پیش از آشنایی با مفهوم تابع در زبان های برنامه نویسی ما همچنان از واژه دستور استفاده خواهیم کرد.

در حالت کلی شیوه نگارش و الگوی بکارگیری این دستور ارزشمند به شکل زیر است :

`range(start, stop, step)`

همانگونه که دیده می شود این دستور دارای سه ورودی به نامهای `start`، `stop` و `step` است که به ترتیب زیر شرح داده می شوند :

start نخستین عددی که دنباله باید از آن آغاز شود را مشخص می کند. مقداردهی به این گزینه اختیاری است و چنانچه برنامه نویس مقدار این گزینه را تعیین نکند دنباله عددی ساخته شده از عدد صفر شروع خواهد شد.

Stop مقدار این گزینه باید یک واحد بیش از آخرین عددی باشد که می خواهید در دنباله تولید شود برای مثال چنانچه این گزینه با عدد ۸ مقدار دهی گردد، آخرین عدد تولید شده در دنباله یک

واحد کمتر ، یعنی عدد ۷ خواهد بود. لازم به گفتن است که مقدار دهی به این گزینه اجباری است.

Step این گزینه گام افزایش start را مشخص می کند. این گزینه اختیاری است و چنانچه برنامه نویس مقدار این گزینه را تعیین نکند مقدار start با گام های یک واحدی افزایش خواهد یافت. برای مثال دستور `range(1,8,2)` دنباله عددی ۱، ۳، ۵، ۷ را تولید می کند. همانگونه که دیده می شود این دنباله عددی از عدد ۱ آغاز شده و با گامهای دو واحدی افزایش می یابد و با عدد ۷ به پایان می رسد.

مثال : دستور `range(6)` دنباله عددی زیر را تولید می کند :

0, 1, 2, 3, 4, 5

در مثال بالا گزینه های start و step مقداردهی نشده اند بنابراین دستور range مقادیر پیش فرض این گزینه ها را استفاده خواهد کرد. یعنی دنباله عددی را با شروع از عدد صفر و با گام های افزایشی یک واحدی که با عدد ۵ پایان می پذیرد خواهد ساخت

مثال :

`range(1,8,3)`

output is 1, 4, 7

مثال :

`range(3,10)`

output is 3, 4, 5, 6, 7, 8, 9

مثال :

`range(3, 20, 2)`

output is 3, 5, 7, 9, 11, 13, 15, 17, 19

عملگرهای عضویت

در پایتون به دو عملگر `in` و `not in` عملگرهای عضویت گفته می شود چرا که این دو عملگر وجود و یا عدم وجود یک مقدار در یک دنباله از عناصر را بررسی می کنند. برای مثال می دانیم که یک رشته دنباله ای از عناصر به نام نویسه^۱ است و یا نتیجه دستور `range()` دنباله ای از اعداد است. بنابراین برای بررسی اینکه آیا نویسه خاصی در یک رشته وجود دارد یا نه؟ می توان از عملگرهای عضویت استفاده کرد. دیگر انواع داده ای زبان برنامه نویسی پایتون که دنباله ای از مقدارها هستند و می توان آنها را به همراه عملگرهای عضویت بکار برد عبارتند از: `set`، `tuple`، `list`، `dict` و `range`

عملگر `in`

با استفاده از عملگر `in` می توان به این پرسش پاسخ داد که آیا یک مقدار مشخص در بین عناصر تشکیل دهنده یک دنباله وجود دارد یا خیر؟ چنانچه عملگر `in` بتواند مقدار داده شده را در دنباله مورد نظر پیدا کند مقدار منطقی `True` را برمی گرداند در غیر این صورت مقدار منطقی `False` برگشت داده خواهد شد.

مثال :

```
string1 = "Nothing in the world is single"
print("w" in string1) # output is True
print("T" in string1) # output is False
```

^۱ character

مثال :

```
FullName = "Mohammad Izanlou"
```

```
if "m" in FullName :
```

```
    print("yes")
```

```
else:
```

```
    print("No")
```

```
# output is yes
```

عملگر not in

هر چند که خروجی منطقی عملگر `not in` عکس عملگر `in` است اما فلسفه وجودی و شیوه کار هر دو یکسان است به بیان ساده تر با استفاده از عملگر `not in` می توان وجود نداشتن یک مقدار مشخص در بین عناصر تشکیل دهنده یک دنباله را بررسی کرد. چنانچه عملگر `not in` نتواند مقدار داده شده را در دنباله مورد نظر پیدا کند مقدار منطقی `True` را برمی گرداند در غیر این صورت مقدار منطقی `False` برگشت داده خواهد شد. برای مثال می دانیم که یک رشته دنباله ای از عناصر به نام نویسه^۱ است. بنابراین برای بررسی عدم وجود یک نویسه خاص در یک رشته می توان از عملگر `not in` استفاده کرد.

مثال :

```
string1 = "Nothing in the world is single"
```

```
print("w" not in string1) # output is False
```

```
print("T" not in string1) # output is True
```

¹ character

دستور input

در بسیاری از برنامه های کاربردی ، داده های مورد نیاز برنامه به وسیله کاربر فراهم می گردد و داد و ستد داده ای کاربر و برنامه مهمترین شکل از برهم کنش و تعامل انسان و نرم افزار است از این رو زبان برنامه نویسی باید دارای ابزارهای لازم برای دریافت ورودی از کاربر باشد. برای دست یابی به این هدف ، مفسر پایتون دستور (به بیان درست تر تابع) `input()` را فراهم آورده است. این دستور برای دریافت ورودی از کاربر استفاده می شود. و حالت کلی نگارش و الگوی بکارگیری آن به شکل زیر است :

```
Variable = input(prompt)
```

Variable نام متغیری است که ورودی گرفته شده از کاربر به صورت یک رشته^۱ در آن ذخیره خواهد شد.

prompt تنها ورودی دستور `input` است و در بردارنده پیامی است که برنامه نویس می خواهد به کاربر نمایش داده شود . مقدار این ورودی باید از نوع رشته باشد. مقداردهی به این ورودی دلخواه است و برنامه نویس می تواند آن را خالی رها کند.

مثال :

```
CountryName = input("نام کشور محل تولد خود را وارد کنید")
```

```
print(CountryName)
```

هنگامی که اجرای کد پایتون به دستور `input()` می رسد، مفسر پایتون با نمایش پیغام مورد نظر برنامه نویس که همان مقدار اختصاص داده شده به گزینه `prompt` است منتظر می ماند تا کاربر، ورودی مورد نظر خود را وارد کند. پس از آنکه کاربر مقدار مورد نظر خود را وارد کرد می تواند با زدن کلید **Enter** پایان ورود اطلاعات را به دستور `input` اعلام کند. پس از این ، دستور `input` مقدار وارد شده توسط کاربر را به صورت یک مقدار از نوع رشته در متغیر `variable` ذخیره می کند. و مفسر پایتون اجرای برنامه را از دستور پس از `input` از سر می گیرد.

¹ string

مثال : قطعه کد زیر با نمایش پیام `Please Enter Your Fist Name:` منتظر ورود یک مقدار از طرف کاربر می ماند ، کاربر نام مورد نظر خود را وارد کرده و با زدن کلید `Enter` در صفحه کلید ، مقدار وارد شده توسط کاربر در متغیر `FirstName` ریخته می شود و با اجرای خط بعدی ، مقدار متغیر `FirstName` توسط دستور `print` در خروجی صفحه نمایش چاپ می شود.

```
FirstName = input("Please Enter Your Fist Name :")
```

```
print(FirstName)
```

نکته بسیار مهمی که لازم است همیشه به یاد داشته باشید این است که دستور `input` ، ورودی وارد شده توسط کاربر را به صورت یک رشته تفسیر می کند. به بیان ساده تر، خروجی دستور `input` یک مقدار از نوع رشته ای است. از این رو داده های عددی وارد شده توسط کاربر باید با استفاده از دستورات تبدیل نوع مناسب ، به نوع عددی مورد نظر تبدیل شوند.

مثال : با فرض اینکه در قطعه کد زیر کاربر ، نخستین ورودی را عدد ۲۵ و ورودی دوم را عدد ۶۰ وارد کند خروجی دستور `print` رشته "۲۵۶۰" است در حالی که مجموع دو عدد ۲۵ و ۶۰ عدد ۸۵ است :

```
num1 = input("Please Enter Your First Number:")
```

```
num2 = input("Please Enter Your Second Number:")
```

```
sum = num1 + num2
```

```
print(sum)
```

مثال : قطعه کد زیر همان کد نوشته شده در مثال قبلی است با این تفاوت که با استفاده از دستور تبدیل نوع ، ورودی های وارد شده توسط کاربر که به ترتیب عددهای ۲۵ و ۶۰ هستند به نوع عددی `int` تبدیل شده اند . بنابراین خروجی دستور `print` عدد ۸۵ خواهد بود.

```
num1 = input("Please Enter Your First Number:")
```

```
num2 = input("Please Enter Your Second Number:")
```

```
sum = int(num1) + int(num2)
print(sum)
```

مثال بالا را می توان به شکل زیر نیز بازنویسی کرد در این مثال با قراردادن دستور `input()` درون دستور `int()` ورودی دریافت شده از کاربر ابتدا به نوع عدد صحیح تبدیل شده و سپس در متغیرهای `num1` و `num2` قرار می گیرند :

```
num1 = int (input("Please Enter Your First Number:"))
num2 = int (input("Please Enter Your Second Number:"))
sum = num1 + num2
print(sum)
```

تبدیل نوع^۱

همانگونه که می دانید نوع داده ، تعیین کننده نوع عملیاتی است که می توان بر روی داده ها انجام داد برای مثال عملیات ریاضی ضرب و تقسیم بر روی نوع داده اعداد صحیح (`int`) و یا اعداد اعشاری (`float`) قابل اجراست اما بر روی نوع داده رشته ای (`str`) قابل اجرا نیست از این رو اجرای دستوری چون `X = "2" × "5"` در پایتون موجب تولید خطا خواهد شد حال چنانچه نوع داده اطلاعات ورودی متفاوت از نوع داده متناسب با عملیات کد نویسی شده در برنامه باشد ، برنامه نویس ناچار است با استفاده از ابزارهایی که پایتون فراهم آورده یک نوع داده را به نوع داده دیگری تبدیل کند . به این نوع تبدیل نوع ، تبدیل نوع صریح گفته می شود برای مثال ، همان گونه که می دانید خروجی دستور `input` یک مقدار از نوع رشته (`str`) است. از این رو داده های عددی که کاربر با استفاده از این دستور به برنامه وارد می کند را باید با بهره گیری از دستورات تبدیل نوع صریح ، به نوع عددی مناسب تبدیل کرد.

¹ Type Casting

دستورات تبدیل نوع صریح در پایتون

Int()

این دستور نوع داده ورودی خود را به نوع داده عدد صحیح (int) تبدیل می‌کند و ورودی این دستور می‌تواند یک عدد صحیح، یک عدد اعشاری و یا رشته ای شامل ارقام ۰ تا ۹ باشد.

مثال : در مثال زیر مفسر پایتون با رسیدن اجرای کد به خط `x = "2"` نوع داده متغیر `x` را از نوع رشته در نظر خواهد گرفت اما با رسیدن اجرا به خط `x = int(x)` نوع داده ای آن به نوع `int` تغییر خواهد یافت. که این موضوع در خروجی دستور `print(type(x))` به روشنی دیده می‌شود.

```
x = "2"
```

```
print(type(x)) #output is <class 'str'>
```

```
x = int(x)
```

```
print(type(x)) #output is <class 'int'>
```

مثال : برنامه زیر دو عدد را از کاربر گرفته و حاصلضرب آن دو را چاپ می‌کند :

```
num1 = input("Please Enter Your First Number:")
```

```
num2 = input("Please Enter Your Second Number:")
```

```
mul = int(num1) * int(num2)
```

```
print(mul)
```

مثال :

```
x = int(1) # x will be 1
```

```
y = int(2.8) # y will be 2
```

```
z = int("51") # z will be 51
```

float()

این دستور نوع داده ورودی خود را به نوع داده عدد اعشاری (float) تبدیل می کند. ورودی این دستور می تواند یک عدد صحیح ، یک عدد اعشاری و یا رشته ای شامل ارقام ۰ تا ۹ باشد.

مثال : در مثال زیر مفسر پایتون با رسیدن اجرای کد به خط `x = "4.5"` نوع داده متغیر `x` را از نوع رشته در نظر خواهد گرفت اما با رسیدن اجرا به خط `x = float(x)` نوع داده ای آن به نوع float تغییر خواهد یافت. که این موضوع در خروجی دستور `print(type(x))` به روشنی دیده می شود.

مثال :

```
x = "4.5"
print(type(x))
#output is <class 'str'>
x = float(x)
print(type(x))
#output is <class 'float'>
```

مثال : برنامه زیر با دریافت شعاع یک دایره ، مساحت و محیط دایره را در خروجی صفحه نمایش چاپ خواهد کرد.

```
pi = 3.14
r = float(input("Please Enter Circle Raduis:"))
c = pi * 2 * r
s = r * r * pi
print(c)
```

`print(s)`

مثال :

`x = float(5) # x will be 5.0`

`y = float (2.8) # y will be 2.8`

`z = float ("17.3") # z will be 17.3`

`str()`

این دستور نوع داده ورودی خود را به نوع داده رشته (`string`) تبدیل می کند. ورودی این دستور می تواند یک عدد صحیح ، یک عدد اعشاری و یا یک رشته باشد.

مثال : در مثال زیر مفسر پایتون با رسیدن اجرای کد به خط `x = 35` نوع داده متغیر `x` را از نوع عدد صحیح (`int`) در نظر خواهد گرفت اما با رسیدن اجرا به خط `x = str(x)` نوع داده ای آن به نوع `string` تغییر خواهد یافت. که این موضوع در خروجی دستور `print(type(x))` به روشنی دیده می شود.

`x = 35`

`print(type(x))`

`#output is <class 'int'>`

`x = str(x)`

`print(type(x))`

`#output is <class 'str'>`

مثال :

`x = str(5) # x will be '5'`

`y = str (2.8) # y will be '2.8'`

`z = str ("ABC") # z will be 'ABC'`

تبدیل نوع ضمنی

در این نوع تبدیل مفسر پایتون به صورت خودکار یک نوع داده را به نوع دیگر تبدیل می کند در این فرآیند نیازی به دخالت برنامه نویس نیست. برای نمونه در مثال زیر متغیر `int_num` از نوع عدد صحیح و متغیر `float_num` از نوع عدد اعشاری است از این رو مفسر پایتون با رسیدن اجرای کد به خط `sum=int_num+float_num` با هدف پیشگیری از ازدست رفتن داده ها ابتدا نوع داده متغیر `int_num` را به نوع `float` تبدیل کرده و سپس حاصل جمع مقدار دو متغیر `float` را در متغیر `sum` ذخیره می کند. با اجرای دستور `print(type(sum))` دیده می شود که نوع داده متغیر `sum` هم `float` است زیرا پایتون با هدف پیشگیری از از دست رفتن داده ها ، همواره نوع داده کوچکتر (در این مثال `int`) را به نوع داده بزرگتر (در این مثال `float`) تبدیل می کند. به بیان دیگر پایتون امن بودن تبدیل ضمنی و پیشگیری از ازدست رفتن داده ها را تضمین می کند. برای درک بهتر مطلب فرض کنید که در مثال زیر مفسر پایتون متغیر `float_num` را به نوع عددی کوچکتر (در این مثال `int`) تبدیل می کرد آنگاه مقدار ذخیره شده در متغیر `float_num` یعنی عدد اعشاری ۱۲٫۸ به نوع عددی صحیح یعنی عدد ۱۲ تبدیل می شد و در نتیجه حاصل جمع ذخیره شده در متغیر `sum` عدد ۳۷ است و این به معنای از دست رفتن بخش اعشاری جواب یعنی ۰٫۸ است.

مثال :

```
int_num = 25
float_num = 12.8
print(type(int_num)) #output is <class 'int' >
print(type(float_num)) #output is <class 'float' >
sum = int_num + float_num
print(type(sum)) #output is <class 'float' >
print(sum) #output is 37.8
```

مثال : در مثال زیر پایتون نمی تواند نوع متغیر x را که یک رشته است به صورت ضمنی به نوع عدد صحیح تبدیل کند و در نتیجه پردازش $z = x + y$ با شکست روبرو شده و یک خطا اعلام خواهد شد.

$x = "5"$

$y = 16$

$z = x + y$

برای جلوگیری از رخ دادن خطا ، باید از روش تبدیل نوع صریح استفاده کرد :

$x = "5"$

$y = 16$

$z = \text{int}(x) + y$

فهرست برخی از مهم ترین دستورات تبدیل نوع صریح در پایتون در جدول زیر آمده است برای آشنایی بیشتر می توانید به منابع موجود در وب مراجعه کنید:

نام دستور	عملکرد دستور
Str()	تبدیل ورودی دستور به نوع رشته ای
int()	تبدیل ورودی دستور به عدد صحیح
float()	تبدیل ورودی دستور به نوع عدد اعشاری
complex()	تبدیل ورودی دستور به نوع عدد مختلط
list()	تبدیل ورودی دستور به نوع list
tuple()	تبدیل ورودی دستور به نوع tuple
range()	تبدیل ورودی دستور به نوع range
dict()	تبدیل ورودی دستور به نوع dict
set()	تبدیل ورودی دستور به نوع set
frozenset()	تبدیل ورودی دستور به نوع frozenset
bool()	تبدیل ورودی دستور به نوع منطقی
bytes()	تبدیل ورودی دستور به نوع bytes
bytearray()	تبدیل ورودی دستور به نوع bytearray
memoryview()	تبدیل ورودی دستور به نوع memoryview

جدول 1-6 دستورات تبدیل نوع صریح در پایتون

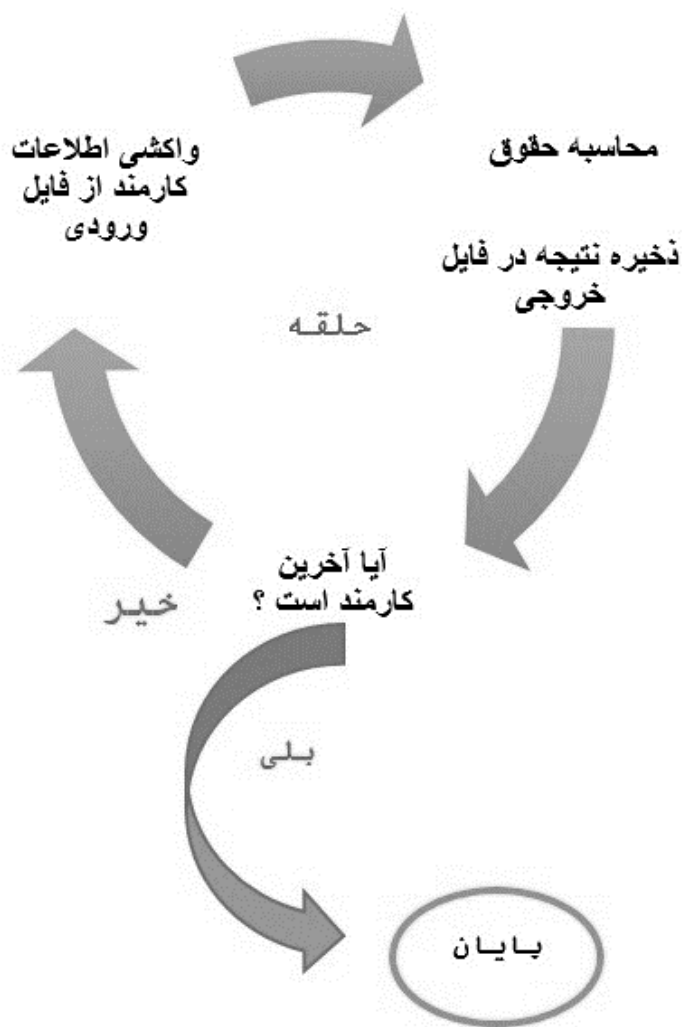
فصل هفتم

حلقه ها

برنامه ساده چاپ اعداد طبیعی کوچکتر از ۴ را در نظر بگیرید ساده ترین راه نوشتن این برنامه ، تکرار دستور Print برای هر یک از اعداد ۰ تا ۴ است تا به ترتیب با چاپ اعداد ۰ ، ۱ ، ۲ ، ۳ ، ۴ به خواسته برنامه نویس برآورده گردد. اما اگر بخواهید برنامه چاپ اعداد طبیعی کوچکتر از ۵۰ را به شیوه گفته شده بنویسید ، به ناچار باید برای چاپ هر یک از اعداد ۰ تا ۴۹ یک دستور `print()` جداگانه بنویسید، ناگفته پیداست که پنجاه بار تکرار دستور `print` روشی غیر منطقی است چراکه این سبک از برنامه نویسی حجم برنامه را بی دلیل افزایش داده و با کاهش خوانایی و سخت تر شدن درک کد ، ویرایش و اصلاح کد در آینده را دشوار می سازد. این دو مثال ساده نیاز به وجود سازوکاری برای برنامه نویسی ساده تر و روان تر کدی که باید بارها و بارها و پشت سرهم اجرا شود را روشن می سازد.

ضرورت وجود این سازوکار دوجندان می شود اگر بدانیم که در بسیاری از الگوریتم های طراحی شده برای حل مسائل دنیای واقعی تکرار و اجرای چندباره کد ، گریزناپذیر است . یک مثال عملی از نیاز به تکرار کد ، محاسبه حقوق و دستمزد کارکنان است. چراکه نرم افزار ناچار به تکرار پردازش محاسبه حقوق و دستمزد برای هر یک از کارکنان به صورت جداگانه است. بنابراین با فرض اینکه تعداد کارکنان یک شرکت فرضی ۲۵۰۰ نفر باشد. نرم افزار مجبور است اجرای سه قطعه کد : واکشی اطلاعات کارمند از فایل ورودی ، محاسبه حقوق و دستمزد و ذخیره نتیجه محاسبه در فایل خروجی را از اولین کارمند شروع و تا رسیدن به آخرین کارمند تکرار کند. یعنی اجرای سه قطعه کد گفته شده ۲۵۰۰ بار تکرار می شود. این مثال نمونه ای از یک تکرار پایان پذیر است زیرا تکرار اجرای کد پس از تعداد اجرای مشخص و یا برقراری یک شرط پایان می پذیرد. در فرهنگ واژگان زبان های

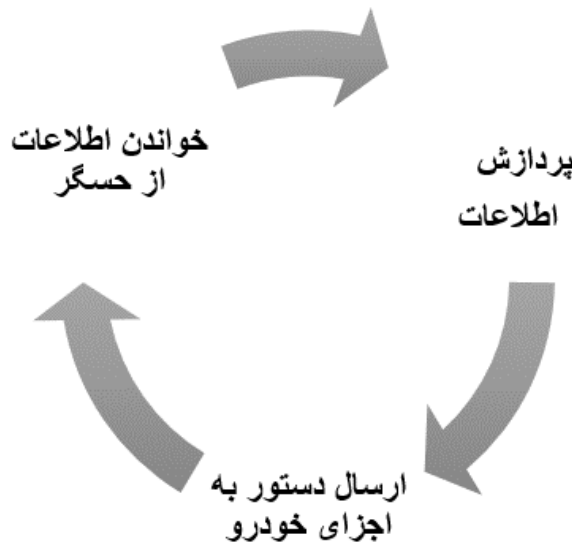
برنامه نویسی به تکرار، حلقه^۱ گفته می‌شود. مفهوم حلقه پایان پذیر در محاسبه حقوق و دستمزد کارکنان در تصویر زیر نمایش داده شده است :



شکل 1-7 مفهوم حلقه

¹ Loop

نوع دیگری از حلقه نیز وجود دارد که به آن حلقه بی نهایت گفته می شود چرا که اجرای کد پس از شروع ، هرگز پایان نخواهد یافت. حلقه بی نهایت بسیار ارزشمند است چرا که در بسیاری از برنامه ها برای چیرگی بر چالشهای دنیای واقعی نیاز است تا یک دستور و یا گروهی از دستورات بی نهایت تکرار شوند برای نمونه رایانه خودرو (ECU) همواره در حال دریافت و پردازش اطلاعات رسیده از حسگر های نصب شده در بخش های گوناگون خودرو و ارسال دستور به اجزای اساسی خودرو است یعنی سه قطعه کد دریافت اطلاعات از حسگرها (برای مثال حسگر اکسیژن) ، پردازش اطلاعات و ارسال دستور (برای مثال تنظیم دقیق ترکیب سوخت و هوا) به اجزای خودرو همیشه در حال تکرار است. مفهوم حلقه بی نهایت یا پایان ناپذیر در تصویر زیر نمایش داده شده است :



شکل 2-7 مفهوم حلقه بی نهایت

در زبان برنامه نویسی پایتون دو دستور ویژه برای پردازش تکرار و پیاده سازی حلقه ها وجود دارد : یکی دستور **for** و دیگری دستور **while** که در فرهنگ واژگان برنامه نویسی بیشتر به نامهای حلقه **for** و حلقه **while** خوانده می شوند. این دو حلقه در ادامه درس با مثالهایی آموزش داده خواهند شد.

حلقه for

در زبان برنامه نویسی پایتون از حلقه **for** بیشتر برای پیمایش دنباله ای از عناصر استفاده می شود از این رو به آن حلقه **for in** هم گفته می شود. لازم به یاد آوری است که بسیاری از انواع داده موجود در پایتون نظیر **string**، **range**، **list**، **tuple**، **dict** و **set** دنباله ای از عناصر هستند برای مثال نوع داده **range** که دنباله ای از اعداد است و یا نوع داده رشته که دنباله ای از نویسه هاست. از این رو محتوای آنها با استفاده از حلقه **for in** به راحتی قابل پیمایش است. برای درک بهتر کدهای نوشته شده با حلقه **for** باید با دستور **range()** و عملگر **in** آشنایی کافی داشته باشید.

حالت کلی نگارش و الگوی بکارگیری حلقه **for** به شکل زیر است :

for Value in Sequence :

Loop Body

Value: یک نام دلخواه برای متغیری است که در هر بار اجرای حلقه ، هر یک از اعضای مجموعه **sequence** در آن قرار می گیرند.

Sequence : یک نوع داده قابل پیمایش است برای مثال یک متغیر رشته ای ، **range** و **list**

Loop Body : بدنه حلقه است که در هربار تکرار حلقه پردازش می شود.

مثال: برنامه ای بنویسید که عددهای طبیعی کوچکتر از ۵۰ را در خروجی صفحه نمایش چاپ کند.

برای درک بهتر مفهوم حلقه در این مثال ساده ، ابتدا الگوریتم آن را می نویسیم و سپس روندنمای^۱ آن را رسم می کنیم :

¹ FlowChart

الگوریتم :

۱- شروع

۲- مقدار ۰ را در متغیر i ذخیره کن

۳- i را چاپ کن

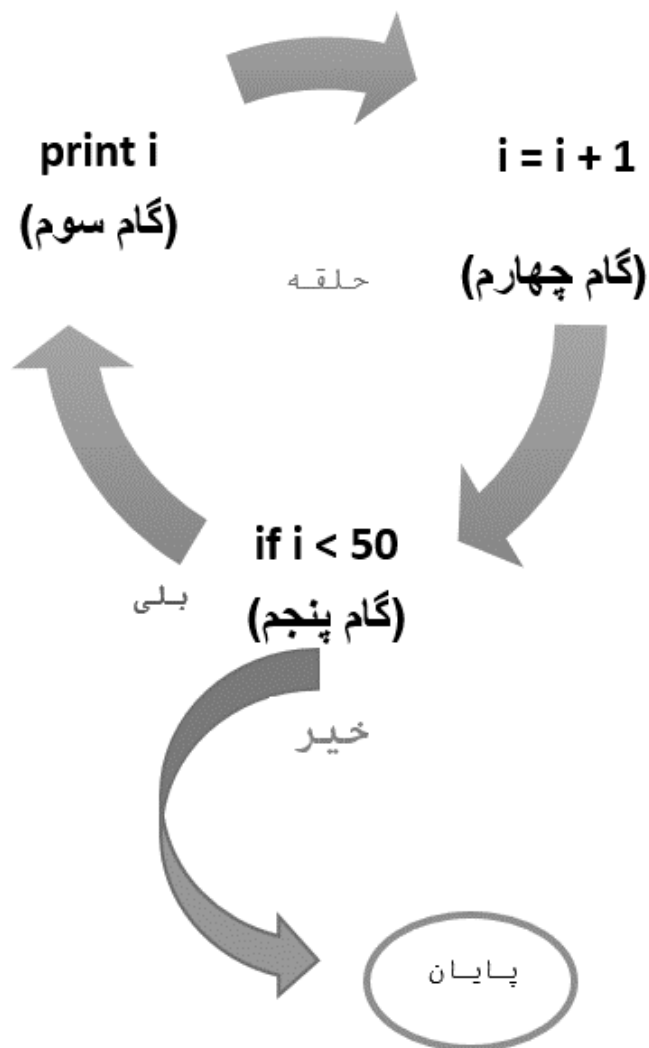
۴- یک واحد به متغیر i اضافه کن

۵- اگر مقدار متغیر i کوچکتر از عدد ۵۰ بود برو به گام سوم

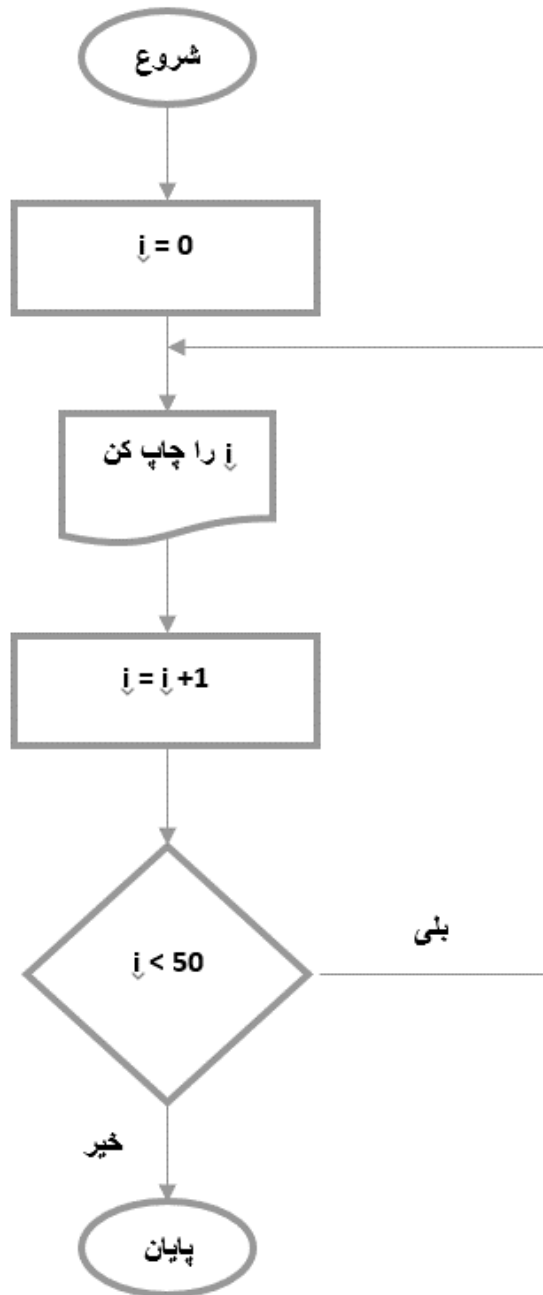
۶- پایان

به گام پنجم دقت کنید تا زمانی که مقدار متغیر i کوچکتر از عدد ۵۰ باشد گامهای سوم و چهارم تکرار خواهند شد به بیان دیگر بین گامهای سوم تا پنجم حلقه ای شکل گرفته که شرط خروج از آن این است که مقدار متغیر i بزرگتر یا مساوی عدد ۵۰ باشد. این مفهوم در روندنمای^۱ زیر به روشنی دیده می شود :

^۱ Flowchart



شکل 3-7 مفهوم حلقه در مثال



شکل 4-7 مفهوم روند نمای الگوریتم حل مسئله

حال با دانسته هایی که از دستور `range()` ، عملگر `in` و الگوی بکارگیری حلقه `for` داریم می توانیم برنامه نویسی این مثال در پایتون را به روش زیر انجام دهیم :

```
for x in range(50):
```

```
    print(x)
```

ابتدا دستور `range(50)` دنباله عددی ۰ تا ۴۹ را تولید می کند، سپس دستور `for` شروع به پیمایش این دنباله عددی می کند یعنی در ابتدا مقدار نخستین عنصر دنباله (عدد ۰) را در متغیر `x` ریخته و پس از آن دستور `print(x)` را اجرا می کند سپس مقدار دومین عدد دنباله (عدد ۱) را در متغیر `x` ریخته و دوباره دستور `print(x)` را اجرا می کند . این تکرار با رسیدن به آخرین عنصر دنباله یعنی عدد ۴۹ و چاپ آن ادامه می یابد و سپس به پایان می رسد.

مثال : برنامه ای بنویسید که اعداد زوج دو رقمی را در خروجی صفحه نمایش چاپ کند.

الگوریتم حل :

۱-شروع

۲- عدد ۱۰ را در متغیر `i` قرار بده

۳- مقدار متغیر `i` را چاپ کن

۴- مقدار متغیر `i` را ۲ واحد افزایش بده

۵- اگر مقدار متغیر `i` کوچکتر از ۱۰۰ بود برو به گام سوم

۶- پایان

برنامه نویسی مسئله با پایتون :

با توجه به دانسته های ما در مورد دستور `range()` و عملگر `in` می توان این حلقه را به صورت زیر نوشت :

```
for x in range(10, 100, 2):
```

```
    print(x)
```

مثال : برنامه ای بنویسید که یک نام خانوادگی را از کاربر گرفته و تک تک نویسه های موجود در آن را در خروجی صفحه نمایش چاپ کند.

می دانیم که هر رشته دنباله ای از یک یا چند نویسه^۱ است از این رو با استفاده از حلقه `for` می توان رشته مورد نظر را پیمایش کرد و با بهره گیری از عملگر `in` هر نویسه را به یک متغیر انتساب داد و سپس با استفاده از دستور `print()` آنرا در خروجی صفحه نمایش چاپ کرد. :

```
LastName = input("please enter your last name :")
```

```
for x in LastName:
```

```
    print(x)
```

مثال : برنامه ای بنویسید که اعداد بین ۱۰۰ تا ۵۰۰ که بر عدد ۲ و یا بر عدد ۹ بخش پذیر هستند را یافته و در خروجی صفحه نمایش چاپ کند :

یادآوری : عددی بر ۲ بخش پذیر است که باقی مانده تقسیم آن عدد بر ۲ صفر باشد. همچنین عددی بر ۹ بخش پذیر است که باقی مانده تقسیم آن عدد بر ۹ صفر باشد.

```
for x in range(100, 501):
```

```
    if (x % 2) == 0 or (x % 9) == 0 :
```

```
        print(x)
```

¹ character

مثال : برنامه ای بنویسید یک رشته را از ورودی گرفته و طول آنرا در خروجی صفحه نمایش چاپ کند. (طول یک رشته ، تعداد نویسه های موجود در آن رشته است)

```
MyString = input("please enter one string :")
```

```
i = 0
```

```
for char in MyString:
```

```
    i+ = 1
```

```
print(i)
```

مثال : برنامه ای بنویسید که مجموع اعداد ۱ تا ۱۰۲۴ را جمع کرده و در خروجی صفحه نمایش چاپ کند.

```
sum = 0
```

```
for num in range(1, 1025):
```

```
    sum = sum + num
```

```
print(sum)
```

حلقه while

شکل کلی نگارش و الگوی بکار گیری این دستور به شکل زیر است :

While شرط حلقه

بدنه حلقه

با استفاده از حلقه while می توان قطعه کد نوشته شده در بدنه آن را تا زمانی که شرط حلقه برقرار است یعنی نتیجه ارزیابی عبارت موجود در بخش شرط حلقه مقدار منطقی True است اجرا کرد. بدیهی است که با False شدن شرط ، حلقه while پایان می پذیرد.

مثال : کد زیر اعداد طبیعی کوچکتر از ۵۰ را چاپ می کند

```
i = 0
```

```
while i < 50 :
```

```
    print(i)
```

```
    i+= 1
```

در برنامه بالا شرط کوچکتر بودن مقدار متغیر i از عدد ۵۰ ($i < 50$) در ابتدای حلقه `while` بررسی می شود چنانچه شرط برقرار بود یعنی نتیجه ارزیابی عبارت مقایسه ای $i < 50$ مقدار منطقی `True` باشد دستور `print(i)` اجرا شده و به مقدار متغیر i یک واحد افزوده می شود و اجرای برنامه به ابتدای حلقه `while` برمی گردد یعنی تکرار بعدی آغاز شده و دوباره شرط $i < 50$ بررسی می شود و این تکرار تا زمانی ادامه می یابد که نتیجه ارزیابی شرط ابتدای حلقه `False` شود.

مثال : برنامه ای بنویسید که مجموع اعداد زوج دو رقمی را محاسبه و چاپ کند:

```
sum = 0
```

```
i = 10
```

```
while i < 100:
```

```
    sum + = i
```

```
    i += 2
```

```
print(sum) # output is 2430
```

مثال : برنامه ای بنویسید که مجموع اعداد سه رقمی که همزمان بر ۷ و ۸ بخش پذیر هستند را محاسبه و چاپ کند.

یادآوری : اگر عدد A بر عدد B بخش پذیر باشد آنگاه باقی مانده تقسیم عدد A بر B برابر صفر است.

```
sum = 0
```

```
i = 100
```

```
while i <= 999:
```

```
    if i % 7 == 0 and i % 8 == 0:
```

```
        sum += i
```

```
    i += 1
```

```
print(sum) # output is : 8512
```

حلقه های تودرتو

گاهی ماهیت مسئله چنان است که برای حل آن به ناچار باید از یک حلقه درون حلقه دیگر استفاده کرد یعنی حلقه دوم در بدنه حلقه نخست جای می گیرد و به ازای هربار تکرار حلقه نخست، تمامی تکرارهای حلقه دوم پردازش می شوند. به این حالت، حلقه های تودرتو گفته می شود. به حلقه نخست حلقه بیرونی^۱ و به حلقه دوم حلقه درونی^۲ گفته می شود. برای مثال فرض کنید دو دنباله عددی A و B همانند زیر تعریف شده باشند، می خواهیم برنامه ای بنویسیم که حاصل ضرب هریک از اعداد موجود در دنباله A در هر یک از اعداد موجود در دنباله B را محاسبه و چاپ کند:

A = 5, 6, 7, 8

B = 4, 5, 6, 7

برای این کار به دو حلقه نیاز داریم، حلقه نخست عناصر موجود در دنباله A را پیمایش می کند و حلقه دوم عناصر موجود در دنباله B را پیمایش خواهد کرد اما با توجه به اینکه هر یک از اعداد موجود در دنباله A باید در تک تک اعداد موجود در دنباله B ضرب شوند از این رو نیاز است تا

¹ outer

² inner

به ازای هر عنصر موجود در A تمامی عناصر موجود در B پیمایش شوند و این یعنی حلقه پیمایش کننده عناصر B باید در بدنه حلقه پیمایش کننده عناصر A جای گیرد:

```
A = range(5,9,1)
```

```
B = range(4,8,1)
```

```
for x in A:
```

```
    for y in B:
```

```
        print(x*y)
```

برنامه بالا را می توان با حلقه while و به شکل زیر بازنویسی کرد:

```
x = 5
```

```
y = 4
```

```
while(x < 9):
```

```
    while(y < 8):
```

```
        print(x*y)
```

```
        y += 1
```

```
    y = 4
```

```
    x += 1
```

مثال : برنامه ای بنویسید که شکل هندسی زیر را با چاپ نویسه ستاره (*) تولید کند.

```

*
**
***
****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****
*****

```

همانگونه که دیده می شود ستاره ها در ۱۲ سطر پشت سرهم قرار گرفته اند و در هر سطر ، تعداد ستاره های چاپ شده برابر با شماره سطر است یعنی در سطر اول یک ستاره ، در سطر دوم دو ستاره و به همین ترتیب ادامه می یابد تا سرانجام در آخرین سطر ۱۲ ستاره چاپ می شود. بنابراین نیاز به وجود دو حلقه است ، حلقه اول دنباله عددی ۱ تا ۱۲ را پیمایش می کند که به معنای پیمایش سطر نخست تا آخرین سطر است ، و حلقه دوم به ازای هر تکرار حلقه اول و به تعداد شماره سطر دستور `print("*")` را اجرا می کند :

```
x = 1
```

```
while(x < 13):
```

```
    y = 1
```

```
    while(y < x + 1):
```

```
        print("*", end = " ")
```

```
        y += 1
```

```
    print("\n")
```

```
    x += 1
```

دستور `print()` پس از چاپ ورودی خود به صورت خودکار به سطر بعدی می‌رود بنابراین در مثال بالا برای جلوگیری از این رفتار پیش فرض یک ورودی جدید به نام `end` و با مقدار رشته خالی (" ") به این دستور داده شده است تا پس از چاپ هر ستاره در سطر جاری بماند و تمامی ستاره های موجود در سطر فعلی در همان سطر چاپ و با یک فاصله خالی از هم جدا شوند. همچنین در مثال بالا دستور `print("\n")` یعنی به سطر بعدی برو ، درحقیقت نویسه `\n` در پایتون بیانگر کد اسکی سطر جدید است.

مثال : برنامه ای بنویسید که دو رشته را از کاربر گرفته و تعداد تکرار هر نویسه موجود در رشته اول را در رشته دوم یافته و چاپ کند.

```
str1 = input("Pleas enter string1:")
str2 = input("Pleas enter string2:")
count = 0
for x in str1:
    for y in str2:
        if x == y:
            count += 1
    print(x, " ", count)
    count = 0
```

یک نمونه از خروجی این برنامه در زیر آمده است :

```
Pleas enter string1:arash
Pleas enter string2:arman
a 2
r 1
a 2
s 0
h 0
```


دستور break

به هنگام خرید از یک فروشگاه بزرگ دیده اید که صندوقدار فروشگاه با استفاده از ابزاری به نام بارکدخوان شناسه یکتای کالای خریداری شده را به نرم افزار وارد می کند و نرم افزار با جستجو و پیمایش فهرست قیمت صدها و گاهی هزاران کالای موجود در فروشگاه که از پیش در پایگاه داده نرم افزار ثبت شده اند، قیمت کالای خریداری شده توسط مشتری را یافته و برای پردازش (محاسبه قیمت کل، اعمال تخفیف، تنظیم فاکتور و ...) در اختیار کد موجود در نرم افزار قرار می دهد با فرض اینکه فروشگاه فرضی مادرای تعداد ۴۰۰۰ کالای گوناگون است و فهرست شناسه تمامی آنها به صورت نامرتب در پایگاه داده نرم افزار ذخیره شده است، در بدترین حالت یعنی حالتی که شناسه کالای مورد نظر در مکان ۴۰۰۰ فهرست ذخیره شده باشد برای جستجوی شناسه آن در فهرست نیاز به اجرای حلقه ای با ۴۰۰۰ بار تکرار است حال فرض کنید شناسه کالای مورد نظر در مکان ۵۰۰ فهرست، ذخیره شده است بنابراین در تکرار پانصدام، نرم افزار کد کالای مورد نظر را یافته و قیمت آن را بدست می آورد. به صورت منطقی پس از این دیگر لازم نیست جستجو ادامه یابد و حلقه باید پایان پذیرد اما می دانیم که در حالت عادی اجرای حلقه `for` زمانی پایان می یابد که پیمایش فهرست به آخرین عنصر موجود در آن برسد و اجرای یک حلقه `while` نیز زمانی پایان خواهد یافت که نتیجه ارزیابی شرط ابتدای حلقه مقدار منطقی `False` باشد. از این رو در مثال بالا بدون وجود راهی برای خروج زود هنگام از حلقه تعداد ۳۵۰۰ تکرار دیگر نیز اجرا خواهند شد بدون آنکه به اجرای آنها نیازی باشد. اهمیت وجود راهی برای خروج زود هنگام از یک حلقه زمانی روشن تر خواهد شد که بدانیم اجرای بیش از نیاز یک حلقه به معنای مصرف منابع باارزشی چون زمان پردازشگر^۱ و فضای حافظه اصلی^۲ رایانه است که خود موجب کندی نرم افزار و کاهش سرعت پاسخگویی به درخواست کاربران از سوی برنامه می شود.

در زبان برنامه نویسی پایتون دستور `break` موجب خروج زود هنگام از حلقه می شود. مفسر پایتون با رسیدن اجرای کد به دستور `break` اجرای حلقه را متوقف می کند و با خارج شدن از حلقه

^۱ CPU

^۲ RAM

دستور پس از حلقه را اجرا می کند. لازم به یادآوری است که در حلقه های تودرتو وجود دستور **break** در بدنه حلقه درونی، تنها موجب خروج از حلقه درونی خواهد شد.

مثال: برنامه زیر اعداد ۱، ۲، ۳ را در خروجی

```
i = 1
while i < 6:
    print(i)
    if i == 3:
        break
    i += 1
```

همان مثال با حلقه **for**:

```
for i in range(۱,۷,۱):
    print(i)
    if i == 3:
        break
```

دستور **continue**

این دستور برخلاف دستور **break** که به اجرای یک حلقه پایان می دهد تنها تکرار فعلی حلقه را پایان داده و اجرا را از تکرار بعدی حلقه از سر می گیرد.

مثال: قطعه کد زیر تنها اعداد فرد موجود در دنباله عددی ۱ تا ۲۰ را چاپ می کند

```
for x in range(1,21,1):
    if x % 2 == 0:
        continue
    print(x)
```

مثال : قطعه کد زیر اعداد 9 تا ۰ را چاپ خواهد کرد اما عدد ۵ چاپ نخواهد شد

```
x = 10
```

```
while x > 0:
```

```
    x = x - 1
```

```
    if x == 5:
```

```
        continue
```

```
    print (x)
```

فصل هشتم

اولویت یا برتری عملگرها

حاصل عبارت ساده ای چون 3×5 و یا $2+3$ همواره روشن و مشخص است و هرکس بدون هیچ گونه ابهامی می تواند بگوید که نتیجه عبارت اولی عدد ۱۵ و نتیجه دومی عدد ۵ است. اما حاصل عبارت $3 \times 2 + 10$ بسته به اینکه ابتدا عملیات ضرب انجام شود و یا عملیات جمع، می تواند دو مقدار متفاوت باشد. چنانچه ابتدا عملیات جمع انجام شود ($2+10=12$) و سپس عملیات ضرب ($3 \times 12=36$) حاصل عبارت $3 \times 2 + 10$ عدد ۳۶ است اما اگر ابتدا عملیات ضرب انجام شود ($3 \times 2=6$) و سپس عملیات جمع ($6+10=16$) حاصل عبارت عدد ۱۶ است. این مثال ساده به روشنی نشان می دهد که در زبانهای برنامه نویسی وجود قوانینی که ترتیب پردازش و اجرای عملگرهای مختلف موجود در یک عبارت را به روشنی و بدون هیچ ابهامی بیان کنند ضرورتی گریزناپذیر است. در فرهنگ واژگان زبانهای برنامه نویسی به این قوانین، اولویت یا برتری عملگرها گفته می شود. به بیان ساده تر برتری عملگرها مجموعه ای از قوانین است که مشخص می کند هنگام ارزیابی یک عبارت معین، کدام عملگر باید زودتر از دیگر عملگرها اجرا شود. برای پیاده سازی این قوانین، زبان برنامه نویسی به هر عملگر و یا دسته ای از عملگرها یک برتری و یا اولویت نسبت می دهد و سپس به هنگام ارزیابی یک عبارت، عملگری که بالاترین اولویت را دارد زودتر از دیگر عملگرها پردازش می کند. برخی از مهمترین قوانین اولویت عملگرها در زبان پایتون عبارتند از:

۱- در زبان پایتون پرانتز دارای بالاترین اولویت است یعنی عبارتی که درون پرانتز قرار گرفته باشد زودتر از دیگر عملگرها ارزیابی خواهد شد. چنانچه در یک عبارت پرانتزهای تودرتو وجود داشته باشند، ترتیب ارزیابی از درونی ترین پرانتز به بیرونی ترین پرانتز است به زبان ساده تر، ابتدا درونی ترین پرانتز ارزیابی خواهد شد.

۲- عملگر انتساب دارای پایین ترین اولویت است.

۳- در بین عملگرهای حسابی ، توان دارای بالاترین اولویت و جمع و تفریق دارای پایین ترین اولویت هستند. چنانچه در یک عبارت بیش از یک عملگر توان به صورت پشت سرهم ظاهر شوند ترتیب ارزیابی آنها از *راست به چپ* خواهد بود.

۴- عملگرهای حسابی ضرب ، تقسیم ، خارج قسمت صحیح و باقی مانده دارای اولویت یکسان هستند. با توجه به اینکه این عملگرها در یک دسته قرار گرفته اند و دارای اولویت یکسان هستند بنابراین در صورت ظاهر شدن همزمان در یک عبارت به ترتیب از *چپ به راست* ارزیابی خواهند شد.

۵- عملگرهای جمع و تفریق دارای اولویت یکسان هستند. با توجه به اینکه این عملگرها در یک دسته قرار گرفته اند و دارای اولویت یکسان هستند بنابراین در صورت ظاهر شدن همزمان در یک عبارت به ترتیب از *چپ به راست* ارزیابی خواهند شد.

۶- عملگرهای مقایسه ای دارای اولویت پایین تری نسبت به عملگرهای ریاضی هستند

۷- عملگرهای منطقی دارای اولویت پایین تری نسبت به عملگرهای مقایسه ای هستند.

۸- عملگر منطقی *and* دارای اولویت بالاتری نسبت به عملگر منطقی *or* می باشد.

ترتیب اولویت یا برتری عملگرهایی که تا کنون با آنها آشنا شده اید در جدول زیر و به ترتیب از بالاترین تا پایین ترین اولویت آمده است :

عملگر	شرح
()	پرانتز
**	به توان رساندن
*, /, //, %	عملگرهای ریاضی ضرب ، تقسیم ، خارج قسمت صحیح و باقی مانده
+, -	جمع و تفریق
<, <=, >, >=, !=, ==	عملگرهای مقایسه ای
and	عملگر منطقی and
or	عملگر منطقی or
=	عملگر انتساب

جدول ۸-۱ اولویت عملگرهای پایتون

مثال : حاصل عبارت $10 - 4 \times 2$ چیست ؟

حل : این عبارت دارای دو عملگر ضرب و تفریق است که با توجه به جدول بالا ، عملگر ضرب دارای اولویت بالاتری نسبت به عملگر تفریق است پس ابتدا باید عملیات ضرب انجام شود و سپس عملیات تفریق در نتیجه :

$$4 \times 2 = 8$$

$$10 - 8 = 2$$

یعنی حاصل عبارت $10 - 4 \times 2$ عدد ۲ است

مثال : حاصل عبارت $(10 - 4) \times 2$ را بدست آورید.

حل : این عبارت دارای سه عملگر ضرب ، تفریق و پرانتز است که با توجه به جدول بالا عملگر پرانتز دارای اولویت بالاتری نسبت به عملگر ضرب است پس ابتدا عبارت درون پرانتز ارزیابی می شود و سپس عملیات ضرب در نتیجه :

$$10-4 = 6$$

$$6 \times 2 = 12$$

یعنی حاصل عبارت $2 \times (10 - 4)$ عدد 12 است.

مثال : دستور `print(5 × 2 // 3)` چه عددی را چاپ خواهد کرد؟

حل : عبارت $5 \times 2 // 3$ دارای دو عملگر ضرب و خارج قسمت صحیح است که با توجه به جدول بالا در یک دسته قرار دارند و در نتیجه دارای اولویت یکسانی هستند بنابراین ترتیب ارزیابی آنها از چپ به راست می باشد در نتیجه :

$$5 \times 2 = 10$$

$$10 // 3 = 3$$

یعنی با اجرای دستور دستور `print(5 × 2 // 3)` عدد ۳ در خروجی صفحه نمایش چاپ خواهد شد.

مثال : دستور `print(5 × (2 // 3))` چه عددی را چاپ خواهد کرد؟

حل : با توجه به جدول بالا عملگر پرانتز دارای اولویت بالاتری نسبت به عملگر ضرب است پس ابتدا عبارت درون پرانتز انجام می شود و سپس عملیات ضرب در نتیجه :

$$2 // 3 = 0$$

$$0 \times 5 = 0$$

یعنی با اجرای دستور دستور `print(5 × (2 // 3))` عدد 0 در خروجی صفحه نمایش چاپ خواهد شد.

مثال : حاصل عبارت $2 ** 3 ** 2$ را بدست آورید.

حل : این عبارت دارای دو عملگر توان است که دارای اولویت یکسانی هستند و هرگاه در یک عبارت بیش از یک عملگر توان به صورت پشت سرهم ظاهر شوند ترتیب ارزیابی آنها از راست به چپ خواهد بود. پس :

$$3^{**}2 = 9$$

$$2^{**}9 = 512$$

یعنی حاصل عبارت $2^{**}3^{**}2$ عدد 512 است.

مثال : حاصل عبارت $2^{**}(3^{**}2)$ را بدست آورید.

حل : پرانتز دارای اولویت بالاتری نسبت به عملگر توان است پس :

$$2^{**}3 = 8$$

$$8^{**}2 = 64$$

یعنی حاصل عبارت $2^{**}(3^{**}2)$ عدد 64 است.

مثال : حاصل عبارت $2 \times 4 - (10 // (2 \times (4 + 6)))$ را بدست آورید.

حل : چنانچه در یک عبارت پرانتزهای تودرتو وجود داشته باشند ، ترتیب ارزیابی از درونی ترین پرانتز به بیرونی ترین پرانتز است به زبان ساده تر، ابتدا درونی ترین پرانتز ارزیابی خواهد شد. پس :

$$(6+4) = 10$$

$$(10 \times 2) = 20$$

$$(20-10) = 10$$

$$(10 // 2) = 5$$

حال با توجه به اینکه اولویت عملگر ضرب بالاتر از عملگر تفریق است :

$$4 \times 2 = 8$$

$$5-8 = -3$$

یعنی حاصل عبارت $2 \times 4 - (10 // (2 \times (4 + 6)))$ عدد -3 است.

مثال : حاصل عبارت $5 \times 3 > 10$ and $4 + 6 == 11$ را بدست آورید.

حل : می دانیم که اولویت عملگرهای حسابی بالاتر از عملگرهای مقایسه ای است پس ابتدا عملگرهای حسابی و از چپ به راست ارزیابی خواهند شد:

$$5 \times 3 = 15$$

$$4 + 6 = 10$$

در نتیجه :

$$15 > 10 \text{ and } 10 == 11$$

از طرفی می دانیم که اولویت عملگرهای مقایسه ای بالاتر از عملگرهای منطقی است پس در گام بعدی عملگرهای مقایسه ای ارزیابی خواهند شد و می دانیم که حاصل ارزیابی عملگرهای مقایسه ای یکی از دو مقدار منطقی True یا False است بنابراین عبارت مقایسه ای $15 > 10$ دارای مقدار True است و عبارت مقایسه ای $10 == 11$ دارای مقدار False است.

پس :

True and False

در آخرین گام عملگر منطقی and ارزیابی می شود :

True and False

که ارزیابی عبارت منطقی True and False مقدار منطقی False خواهد بود. پس حاصل عبارت $5 \times 3 > 10$ and $4 + 6 == 11$ مقدار منطقی False است.

مثال : خروجی قطعه کد زیر چیست ؟

$$X = (5 + 3) \times 2 ** 2$$

print(X)

حل : با توجه به اولویت عملگرها اجرای آنها به ترتیب زیر است :

$5+3 = 8$

$2**2 = 4$

$X = 8 \times 4$

$X = 32$

پس مفسر پایتون با رسیدن اجرای برنامه به دستور `print(X)` عدد ۳۲ را در خروجی صفحه نمایش چاپ خواهد کرد.

مثال : خروجی برنامه زیر چیست؟

`x = 3 + 8 × 2** 3`

`print(x)`

`y = (4 × 2) + 6 / 3 - 2`

`print(y)`

جواب : دستور `print(x)` عدد ۶۷ و دستور `print(y)` عدد ۸ را در خروجی صفحه نمایش چاپ خواهند کرد.

مثال : خروجی برنامه زیر چیست ؟

`name = "Arash"`

`age = 12`

`if name == "Arash" or name == "Arman" and age >= 11 :`

`print("Hello! Welcome.")`

`else :`

`print("Good Bye!!")`

حل : عملگرهای مقایسه ای دارای اولویت بالاتری نسبت به عملگرهای منطقی هستند و اولویت عملگر منطقی `and` بالاتر از اولویت عملگر منطقی `or` است پس در عبارت موجود در بخش شرط دستور `if` یعنی عبارت `name == "Arash" or name == "Arman" and age >= 11`

ابتدا عملگرهای مقایسه ای ارزیابی می شوند و می دانیم که نتیجه ارزیابی یک عبارت مقایسه ای یک از دو مقدار منطقی True یا False است حال براساس داده‌های برنامه می دانیم که:

```
name = "Arash"
```

```
age = 12
```

پس نتیجه ارزیابی عبارت `name == "Arash"` مقدار منطقی True و نتیجه ارزیابی عبارت `age >= 11` مقدار منطقی False است همچنین نتیجه ارزیابی عبارت `name == "Arman"` مقدار منطقی True است پس تا اینجا عبارت

```
name == "Arash" or name == "Arman" and age >= 11
```

به صورت زیر ارزیابی می شود:

True or False and True

در گام بعدی و با توجه به اینکه عملگر `and` دارای اولویت بالاتری نسبت به عملگر `or` است پس ابتدا عملگر `and` ارزیابی می شود پس عبارت `True or False and True` به صورت زیر ارزیابی خواهد شد :

False and True == False

True or False == True

بنابراین نتیجه پایانی ارزیابی عبارت

```
name == "Arash" or name == "Arman" and age >= 11
```

مقدار منطقی True است بنابراین دستور `print("Hello! Welcome.")` اجرا خواهد شد.

فصل نهم

نوع داده list

List یا همان فهرست ، یکی از پرکاربردترین ساختمان داده های موجود در پایتون است و کاربرد اصلی آن ذخیره چندین مقدار مختلف در یک متغیر واحد و دسترسی به همه آنها تنها با یک نام مشخص است. به زبان ساده تر به جای آنکه برای ذخیره موقت هر مقدار، یک متغیر جداگانه تعریف شود ، تنها یک متغیر تعریف و تمامی مقادیر مورد نیاز در آن ذخیره و از آن بازیابی می شوند. برای مثال برای محاسبه میانگین قد دانش آموزان یک مدرسه ، نیازی به تعریف متغیرهای جداگانه برای ذخیره قد هر دانش آموز نیست و می توان با تعریف یک list و ذخیره قد تمامی دانش آموزان مدرسه در آن پردازش مورد نیاز برای محاسبه میانگین قد را بر روی list انجام داد.

در حقیقت فهرست یکی از چهار نوع داده موجود در پایتون است که ذخیره مجموعه ای از داده ها در یک متغیر واحد را ممکن می سازند. و هم زمان ابزارهای لازم برای جستجو ، افزودن ، حذف و به روزرسانی داده های ذخیره شده در آن را فراهم می آورند این چهار نوع عبارتند از :

فهرست (list)

چندتایی (tuple)

مجموعه (set)

واژه نامه (dictionary)

در پایتون یک فهرست به دو روش زیر ساخته می شود:

۱- مقدار دهی مستقیم به یک متغیر

در این روش مقادیر مورد نظر که با نویسه , از هم جدا می شوند درون یک جفت علامت [] قرار می گیرند و سپس با استفاده از عملگر جایگزینی = در متغیر مورد نظر ذخیره می شوند.

مثال : فهرستی از شرکت های فعال در صنعت خودروسازی که دارای چهار مقدار است.

```
cars = ["Toyota", "Peugeot", "", "Mercedes Benz", "Hundai"]
print(cars)
```

مثال : فهرستی از نمرات درس ریاضی دانش آموزان یک کلاس که دارای ۱۰ عنصر است.

```
grades = [18,18.30,12,17.5,16,19,17,14.5,13,15]
print(grades)
```

۲- استفاده از سازنده^۱ کلاس List

مثال : فهرستی از نام دانش آموزان یک کلاس

```
student=list(("Arash","Arman","Mina","Darush","Sepideh"))
print(student)
```

مثال : فهرستی از نمرات درس ریاضی تعدادی از دانشجویان

```
grades=list((18,18.30,12,17.5,16,19,17,14.5,13,15))
print(grades)
```

فهرست ها دارای چهار ویژگی زیر هستند :

۱- عناصر درون یک فهرست دارای ترتیب^۲ مشخص هستند نه به این معنی که عناصر از کوچکترین مقدار به بزرگترین مقدار و یا برعکس مرتب شده اند بلکه به این معنی که عناصر درون فهرست دارای ترتیب قرار گیری مشخصی هستند به زبان ساده تر محل قرار گرفتن عناصر یک فهرست بر اساس ترتیب ورود آنها به فهرست و یا به هنگام تعریف و مقدار دهی اولیه به آن تعیین می شود . به عددی که نشان دهنده محل قرار گرفتن هر عنصر درون فهرست است شاخص^۳ گفته می شود. شاخص گذاری عناصر موجود در یک

^۱ constructor

^۲ Ordered

^۳ index

فهرست از عدد صفر شروع می شود یعنی نخستین عنصر دارای شاخص صفر ، دومین عنصر دارای شاخص یک و به همین ترتیب آخرین عنصر دارای شاخص $n-1$ است که n طول فهرست یا همان تعداد عناصر موجود در فهرست است. ناگفته نماند که کلاس `list` دارای شگردهایی^۱ است که می توانند ترتیب قرار گرفتن عناصر در یک فهرست را به هم بزنند اما در حالت کلی یک فهرست دارای ویژگی ترتیب است.

۲- فهرست یک شی قابل تغییر^۲ است به این معنی که پس از تعریف یک فهرست و مقدار دهی اولیه به آن می توان عناصر موجود در آن را حذف کرد ، تغییر داد و یا عناصر جدیدی به آن افزود.

۳- عناصر درون یک فهرست می توانند تکراری باشند. توانایی فهرست ها در پذیرش مقادیر تکراری مرهون شاخص گذاری عناصر موجود در آن است تا زمانی که هر دو مقدار یکسان شاخص متفاوتی داشته باشند هیچ مشکلی پیش نخواهد آمد.

مثال : فهرست زیر دارای دو عنصر با مقدار تکراری "Blue" و شاخص های متفاوت .
و ۴ است

```
colorList = ["Blue", "Red", "Purple", "Green", "Blue"]
```

```
print(colorList)
```

۴- هر یک از عناصر موجود در یک فهرست می توانند نوع داده متفاوتی داشته باشند

مثال : در فهرست `myFavoritWords` نخستین عنصر از نوع رشته ، دومین عنصر یک عدد صحیح و سومین عنصر یک عدد اعشاری است.

```
myFavoritWords = ["Hundai", ۲۰,۵۶,۵, "Iran"]
```

¹ Methods

² Changeable

```
print(myFavoritWords)
```

روش دستیابی به عناصر یک فهرست

همانگونه که گفته شد عناصر موجود در یک فهرست شاخص گذاری می شوند یعنی محل قرارگیری عناصر موجود در آن به ترتیب و با شروع از عدد صفر شماره گذاری می شود بنابراین می توان با قراردادن شاخص هر عنصر درون عملگر انتخاب (جفت علامت [] پس از نام فهرست) ، به مقدار آن دست یافت برای نمونه در مثال زیر `student[0]` نخستین عنصر فهرست یعنی "Arash" و `student[3]` چهارمین عنصر فهرست یعنی "Bahram" را بازیابی می کند. دقت کنید که شاخص گذاری عناصر در یک فهرست از صفر شروع می شود.

مثال :

```
student = ["Arash", "Arman", "Mina", "Bahram", "Sepideh"]
print(student[0])
print(student[1])
print(student[2])
print(student[3])
print(student[4])
```

فهرست ها از شاخص گذاری منفی هم پشتیبانی می کنند بدین معنی که آخرین عنصر یک فهرست یعنی نخستین عنصر از انتهای فهرست دارای شاخص 1- دومین عنصر از انتهای فهرست دارای شاخص 2- و به همین ترتیب تا نخستین عنصر که دارای شاخص n- است که n تعداد کل عناصر موجود در فهرست است.

```
student = ["Arash", "Arman", "Mina", "Bahram", "Sepideh"]
```

شاخص گذاری منفی	-۵	-۴	-۳	-۲	-۱
شاخص گذاری مثبت	0	1	2	3	4
Student List	Arash	Arman	Mina	Bahram	Sepideh

مثال: دستور `print(thislist[-1])` در کد زیر عنصر "Third" را چاپ خواهد کرد.

```
thislist=["First", "Second", "Third"]
print(thislist[-1])
```

دیگر روش دستیابی به عناصر موجود در یک فهرست استفاده از حلقه `for` به همراه عملگر `in` است.

مثال :

```
student = ["Arash", "Arman", "Mina", "Darush", "Sepideh"]
for itm in student:
    print(itm)
```

طول فهرست

به تعداد عناصر موجود در یک فهرست طول فهرست گفته می شود برای بدست آوردن طول یک فهرست می توانید از تابع `len()` استفاده کنید.

مثال :

```
grades = [18,18.30,12,17.5,16,19,17,14.5,13,15]
length = len(grades)
print(length) #OutPut is 10
```

مثال : کد زیر عناصر موجود در فهرست `student` را از انتها به ابتدا چاپ می کند. برای درک بهتر کد زیر دوباره یادآوری می شود که فهرست ها از شاخص گذاری منفی عناصر پشتیبانی می کنند بدین معنی که آخرین عنصر یک فهرست دارای شاخص 1- دومین عنصر از انتهای فهرست دارای شاخص 2- و به همین ترتیب تا نخستین عنصر از ابتدای فهرست که دارای شاخص n- است که n تعداد کل عناصر موجود در فهرست است.


```
student = list(("Arash" , "Arman", "Mina" ,"Bahram", "Sepideh"))
end = -1
start = -len(student)
while end >= start:
    print(student[end])
    end += -1
```

انتخاب محدوده معینی از یک فهرست

می توان محدوده ای از عناصر موجود در یک فهرست را انتخاب کرد برای این کار باید محدوده عناصر مورد نظر را به صورت Start:End درون عملگر انتخاب (جفت علامت []) پس از نام فهرست قرارداد یعنی محدوده مورد نظر را باید به صورت [Start:End] مشخص کرد در این حالت عناصری که دارای شاخص Start تا End-1 هستند انتخاب می شوند. بنابراین به یاد داشته باشید که شاخص End را یک واحد بیش از شاخص آخرین عنصر مورد نظر انتخاب کنید. مقدار پیش فرض شاخص Start صفر است پس اگر مقدار آن خالی رها شود محدوده انتخاب از نخستین عنصر موجود در فهرست شروع می شود همچنین مقدار مقدار پیش فرض شاخص End برابر طول فهرست است پس چنانچه مقدار آن خالی رها شود محدوده انتخاب از شاخص Start آغاز و تا آخرین عنصر موجود در فهرست که دارای شاخص n-1 است ادامه خواهد یافت. (n طول فهرست است)

مثال :

```
thislist = ["Arash" , "Arman", "Mina" ,"Bahram", "Sepideh"]
print(thislist[2:4])
```

خروجی کد بالا ["Mina" ,"Bahram"] است

مثال:

```
list1 = [10,20,30,40,50]
list2 = list1[:]
```

```
print(list2)
```

#output is : [10, 20, 30, 40, 50]

کد بالا تمامی عناصر موجود در list1 را انتخاب و در list2 قرار می دهد

مثال :

```
thislist = ["Arash", "Arman", "Mina", "Bahram", "Sepideh"]
```

```
partList = thislist[:4]
```

```
print(partList)
```

کد بالا عناصر "Arash", "Arman", "Mina", "Bahram" را چاپ می کند

مثال : کد زیر عناصر "Mina", "Bahram", "Sepideh" را چاپ خواهد کرد

```
thislist = ["Arash", "Arman", "Mina", "Bahram", "Sepideh"]
```

```
print(thislist[2:])
```

مثال : کد زیر عناصر "Mina", "Bahram" را چاپ می کند

```
thislist = ["Arash", "Arman", "Mina", "Bahram", "Sepideh"]
```

```
partList = thislist[-3:-1]
```

```
print(partList)
```

برای اطمینان از موجود بودن عنصر مورد نظر در یک فهرست مشخص می توانید از دستور if به همراه عملگر in استفاده کنید.

مثال :

```
thislist = ["Arash" , "Arman" , "Mina" ,"Bahram" , "Sepideh"]
```

```
if "Arman" in thislist:
```

```
    print("Arman is in the List")
```

برای اطمینان از اینکه عنصر مورد نظر در فهرست وجود ندارد می توانید از دستور if به همراه عملگر not in استفاده کنید

مثال :

```
thislist = ["Arash" , "Arman" , "Mina" ,"Bahram" , "Sepideh"]
```

```
if "Amir" not in thislist:
```

```
    print("Amir is not in the List")
```

به روز رسانی فهرست

چنانکه گفته شد عناصر موجود در یک فهرست با استفاده از شاخص آنها دستیابی می شوند از این رو با دانستن شاخص عنصری که می خواهیم تغییر دهیم می توان به راحتی به آن دست یافت و مقدارش را تغییر داد. برای نمونه فهرست averages که به صورت زیر تعریف و مقداردهی شده است در بردارنده معدل تعدادی از دانشجویان دوره کارشناسی رشته حسابداری است در این فهرست معدل نخستین دانشجو که دارای شاخص صفر است ۱۹ و معدل سومین دانشجو که دارای شاخص ۲ است ۱۶ می باشد دقت کنید که شاخص عناصر موجود در یک فهرست به ترتیب قرار گرفتن آنها در فهرست و از صفر شروع می شود حال فرض کنید که معدل اولین و سومین دانشجو نیاز به تغییر دارد برای انجام این کار می توانید همانند زیر کد نویسی کنید :

```
averages = [19, 18, 16 , 18.8, 17, 15.5, 19.8 , 20 , 14]
```

```
averages[0] = 19.7
```

```
averages[2] = 17
```

```
print(averages)
```

```
#OutPut [19.7, 18, 17, 18.8, 17, 15.5, 19.8, 20, 14]
```

مثال : فهرست زیر حاوی حقوق دریافتی کارگران یک کارگاه صنعتی در پایان بهمن ماه سال ۱۴۰۱ با واحد پولی تومان است. مدیر کارگاه می خواهد به کارگرانی که حقوق ماهانه آنها کمتر از ۵ میلیون تومان است ۱۵ درصد اضافه حقوق پرداخت کند برنامه ای بنویسید که فهرست اولیه را بر اساس نظر مدیر کارگاه تغییر و فهرست جدید حقوق را در خروجی صفحه نمایش چاپ کند.

```
salary = [5500000, 4300000, 5600000, 7000000, 4400000, 5700000, 4000000]
```

حل :

```
salary = [ 5500000, 4300000, 5600000,7000000, 4400000,5700000,4000000]
```

```
print(salary)
```

```
i = 0
```

```
while i < len(salary):
```

```
    if salary[i] < 5000000:
```

```
        inc = salary[i] * 0.15
```

```
        salary[i] += inc
```

```
    i += 1
```

```
print(salary)
```

مثال : برنامه زیر عناصر دو فهرست به نام های list1 و list2 را با هم جابجا می کند

```
list1 = ["Arash", "Arman", "Farhad", "Rostam"]
```

```
list2 = ["Sepideh", "Dariush", "Faramarz", "Nilufar"]
```

```
print("list1 Before Change: ", list1)
```

```
print("Listis Before Change: ", list2)
```

```
print("=====")
```

```
i = 0
while i < len(list1):
    temp = list1[i]
    list1[i] = list2[i]
    list2[i] = temp
    i += 1
print("list1 After Change: ", list1)
print("Listis After Change: ", list2)
```

بروزرسانی یک محدوده از عناصر موجود در فهرست

برای تغییر یک محدوده مشخص از عناصر موجود در یک فهرست، در ابتدا باید فهرست جدیدی که دربردارنده مقادیر جدید مورد نظر است را بسازید و با استفاده از عملگر جایگزینی یا همان انتساب (=) آن را با محدوده مورد نظر از فهرستی که می خواهید تغییر دهید جایگزین کنید.

مثال: کد زیر عناصر موجود در فهرست second را جایگزین شش عنصر نخست فهرست first می کند.

```
first = [25, 12, 10, 11, 14, 12, 13, 15, 16, 10, 8, 7]
second = [23, 14, 12, 10, 8, 9]
print(first)
first[0:6] = second
print(first)
```

همانگونه که گفته شد ساختمان داده list ابزارهای لازم برای جستجو، افزودن، حذف، نسخه برداری و بروزرسانی داده های ذخیره شده در فهرست ها را فراهم می آورد. برای این منظور در

کلاس `list` شگردهای^۱ گوناگونی پیش بینی شده است که در ادامه به برخی از مهم ترین آنها می پردازیم :

شگرد(`insert()`)

با استفاده از این شگرد^۲ می توان یک عنصر را در مکان دلخواهی از فهرست وارد کرد. الگوی کلی نگارش و بکارگیری این شگرد همانند زیر است :

```
list.insert(index, element)
```

این شگرد دارای دو ورودی ظاهری^۳ است : ورودی نخست یا `index` همان شاخصی است که مکان قرار گیری عنصر مورد نظر در فهرست را مشخص می کند. دومین ورودی یا `element` همان عنصری است که می خواهید به فهرست اضافه شود. این ورودی می تواند یک شی از هر نوع داده ای باشد.

مثال : در کد زیر مقدار "Sina" در مکان اشاره شده توسط شاخص ۱ در فهرست `player` درج می شود ، با توجه به اینکه شماره گذاری عناصر یک فهرست از صفر شروع می شود بنابراین مقدار "Sina" دومین عنصر موجود در فهرست `player` است.

```
Player = ["Arash", "Arman", "Kaveh"]
```

```
Player.insert(1, "Sina")
```

```
print(Player) #OutPut is ['Arash', 'Sina', 'Arman', 'Kaveh']
```

مثال : در کد زیر فهرست `ClassAStudents` در مکان اشاره شده توسط شاخص ۲ در فهرست `AllStudents` درج می شود ، با توجه به اینکه شماره گذاری عناصر یک فهرست از صفر شروع می شود بنا براین فهرست `ClassAStudents` سومین عنصر موجود در فهرست `AllStudents` است.

¹ Methods

² Method

³ Parameter

```
AllStudents = ["Farhad", "Neda", "Mahin", "Omid"]  
ClassAStudents = ["Simin", "Maryam", "Sara"]  
AllStudents.insert(2, ClassAStudents)  
print('Updated list:', AllStudents)  
  
#OutPut is : Updated list: ['Farhad','Neda',['Simin','Maryam','Sara'],  
'Mahin', 'Omid']
```

شگرد append()

با استفاده از این شگرد می توان یک عنصر را در انتهای فهرست وارد کرد. به زبان ساده تر کار این شگرد درج یک عنصر در انتهای فهرست است. الگوی کلی نگارش و بکارگیری این شگرد همانند زیر است :

```
list.append(element)
```

شگرد `append()` تنها دارای یک ورودی ظاهری^۱ به نام `element` است که همان عنصری است که می خواهید به انتهای فهرست اضافه شود. این ورودی می تواند یک شی از هر نوع داده ای باشد.

مثال :

```
fruits = ["apple", "banana", "cherry"]  
fruits.append("orange")  
print(fruits)
```

¹ Parameter

```
#output : ['apple', 'banana', 'cherry', 'orange']
```

شگرد extend()

با استفاده از شگرد extend() می توان تمامی عناصر موجود در یک فهرست دیگر و یا هر نوع داده قابل پیمایشی چون set ، tuple و dictionary را در انتهای فهرست مورد نظر وارد کرد. به زبان ساده تر کار این شگرد درج تمامی عناصر موجود در یک شی قابل شمارش در انتهای یک فهرست است. الگوی کلی نگارش و بکارگیری این شگرد به شکل زیر است :

```
list.extend(iterable)
```

شگرد extend() تنها دارای یک ورودی ظاهری به نام iterable است که همان شی قابل شمارشی است که می خواهید عناصر موجود در آن در انتهای فهرست مورد نظر درج گردند. این ورودی می تواند یک شی از نوع list ، tuple ، set ، dictionary و یا هر نوع داده قابل پیمایش دیگری باشد.

مثال :

```
A = ["Farhad", "Neda", "Mahin", "Omid"]
B = ["Simin", "Maryam", "Sara"]
A.extend(B)
print(A)
#output : ['Farhad', 'Neda', 'Mahin', 'Omid', 'Simin', 'Maryam', 'Sara']
```

مثال : در کد زیر یک شی از نوع داده مجموعه (set) به نام frutSet به انتهای فهرست fruits اضافه شده است. با نوع داده set در فصل های بعدی این کتاب آشنا خواهید شد.

```
fruits = ["apple", "banana", "cherry"]
frutSet = ("kiwi", "orange")
```



```
fruits.extend(fruitsSet)
print(fruits)
#output : ['apple', 'banana', 'cherry', 'kiwi', 'orange']
```

حذف عناصر یک فهرست

برای حذف یک عنصر دلخواه از یک فهرست مشخص سه روش وجود دارد :

۱- استفاده از شگرد `remove()`

۲- استفاده از شگرد `pop()`

۳- بکارگیری دستور `del`

شگرد `remove()` برای حذف یک عنصر بر اساس مقدار آن بکار می رود به زبان ساده تر ورودی این شگرد مقداری است که می خواهید از فهرست حذف گردد برای نمونه در مثال زیر می خواهیم نخستین عنصری که دارای مقدار ۱۸ است از فهرست `grades` حذف شود :

```
grades = [20, 18.5, 15, 17, 18, 16, 19, 18, 15]
print(grades)
grades.remove(18)
print(grades)
#output [20, 18.5, 15, 17, 18, 16, 19, 18, 15]
#output [20, 18.5, 15, 17, 16, 19, 18, 15]
```

شگرد `pop()` برای حذف یک عنصر از فهرست بر اساس شاخص آن بکار می رود به زبان ساده تر ورودی این شگرد شاخص عنصری است که می خواهید از فهرست حذف شود برای نمونه در مثال زیر می خواهیم چهارمین عنصر فهرست حذف گردد با توجه به اینکه شاخص گذاری عناصر موجود در یک فهرست از صفر شروع می شود پس دستور `grades.pop(3)` منظور ما را برآورده می سازد :

```
grades = [20, 18.5, 15, 17, 18, 16, 19, 18, 15]
```

```
print(grades)
grades.pop(3)
print(grades)
#output [20, 18.5, 15, 18, 16, 19, 18, 15]
```

چنانچه شاخصی برای شگرد `pop()` مشخص نشود این شگرد آخرین عنصر موجود در فهرست را حذف خواهد کرد :

```
Customers = ["arash" , "Arman", "Farhad", "Kaveh"]
Customers.pop()
print(Customers)
#output ['arash', 'Arman', 'Farhad']
```

دستور `del` هم برای حذف یک عنصر از فهرست بر اساس شاخص آن بکار می رود :

```
Customers = ["arash" , "Arman", "Farhad", "Kaveh"]
del Customers[3]
print(Customers)
#output ['arash', 'Arman', 'Farhad']
```

دستور `del` توانایی حذف کل یک فهرست را دارد و با استفاده از آن می توان فهرست مورد نظر را حذف کرد :

```
Customers = ["arash" , "Arman", "Farhad", "Kaveh"]
del Customers
print(Customers)
#output Error name 'Customers' is not defined
```

حذف تمامی عناصر یک فهرست

شگرد `clear()` برای خالی کردن یک فهرست از عناصر موجود در آن استفاده می شود این شگرد تمامی عناصر موجود در یک فهرست را حذف می کند اما خود فهرست همچنان باقی می ماند و می توان دوباره عناصر جدیدی در آن ذخیره کرد.

```
Customers = ["arash" , "Arman", "Farhad", "Kaveh"]
Customers.clear()
print(Customers)
#output []
```

پیمایش فهرست ها

پیمایش یک فهرست به معنای حرکت در میان عناصر موجود در آن و خواندن مقدار هر یک از آنها به ترتیب ملاقات است. پیمایش می تواند از یک نقطه دلخواه آغاز و در یک نقطه دلخواه دیگر به پایان برسد. در بسیاری از کاربردها نخستین عنصر فهرست نقطه آغاز پیمایش است و آخرین عنصر موجود در فهرست نقطه پایان. جهت پیمایش می تواند از ابتدا به انتهای فهرست و یا برعکس باشد. پیمایش یک فهرست با استفاده از حلقه ها انجام می شود:

مثال: پیمایش فهرست Customers با استفاده از حلقه `for`

```
Customers = ["arash" , "Arman", "Farhad", "Kaveh"]
for itm in Customers:
    print(itm)
```

مثال : پیمایش فهرست Customers با استفاده از حلقه `while`

```
Customers = ["arash" , "Arman", "Farhad", "Kaveh"]
i=0
while i < len(Customers):
    print(Customers[i])
    i += 1
```

مثال : پیمایش فهرست Customers با بهره گیری از حلقه for ، مفهوم شاخص و دستور range())

```
Customers = ["arash" , "Arman", "Farhad", "Kaveh"]
for i in range(len(Customers)):
    print(Customers[i])
```

ساده سازی پیمایش فهرست با شیوه فهرست برگزیده

فهرست برگزیده^۱ یک شیوه ساده در پایتون برای پالایش یک فهرست بر اساس برخی از معیارهای مورد نظر برنامه نویس و ساخت یک فهرست جدید است به زبان ساده تر راهی ساده برای ساخت یک فهرست جدید از روی یک فهرست موجود بر مبنای معیارهای دلخواه است. برای مثال می خواهیم از فهرستی به نام StudentsGrades که دربردارنده نمرات درس ریاضی تعدادی از دانش آموزان است فهرست جدیدی بدست آوریم که دربردارنده نمره های بالاتر از ۱۷ است برای حل این مسئله در ابتدا از روش قدیمی پیمایش استفاده می کنیم :

```
StudentsGrades=[15,16.5,17.8,19,18,20,14,16,13,19.5]
print(StudentsGrades)
NewGrades=[]
for x in StudentsGrades:
    if x > 17:
        NewGrades.append(x)
print(NewGrades)
```

با روش فهرست برگزیده برنامه حل مسئله به صورت زیر نوشته می شود :

```
StudentsGrades=[15,16.5,17.8,19,18,20,14,16,13,19.5]
print(StudentsGrades)
```

¹comprehension

```
NewGrades=[x for x in StudentsGrades if x > 17 ]  
print(NewGrades)
```

همانگونه که دیده می شود فهرست برگزیده موجب کوتاه تر شدن کد و فشرده نویسی آن می شود و در نتیجه خوانایی برنامه افزایش می یابد.

مثال : از فهرست `country` که در بردارنده نام برخی از کشورهاست کشورهایی را انتخاب کنید که در نام آنها نویسه `a` وجود نداشته باشد.

```
country = ["Iran","Canada","China","Egypt","Greece","India", "Italy"]  
print(country)  
Newcountry = [x for x in country if "a" not in x ]  
print(Newcountry)  
#output is ['Egypt', 'Greece']
```

مثال : فهرست `country` در بردارنده نام تعدادی از کشورهاست برنامه ای بنویسید که تمامی حروف موجود در نام هر کشور را به حروف بزرگ تبدیل و نتیجه را در لیست جدیدی به نام `Newcountry` ذخیره کند این مسئله را هم به روش حلقه عادی و هم به روش فهرست برگزیده برنامه نویسی کنید:

برنامه نویسی راه حل مسئله به روش پیمایش حلقه

```
country = ["Iran", "Canada","China","Egypt","Greece","India","Italy"]  
print(country)  
Newcountry = []  
for x in country:  
    Newcountry.append(x.upper())  
print(Newcountry)
```

```
#output ['IRAN', 'CANADA', 'CHINA', 'EGYPT', 'GREECE', 'INDIA', 'ITALY']
```

برنامه نویسی راه حل مسئله به روش فهرست برگزیده :

```
country=["Iran", "Canada","China","Egypt", "Greece","India","Italy"]
```

```
print(country)
```

```
Newcountry = [x.upper() for x in country]
```

```
print(Newcountry)
```

```
#output ['IRAN','CANADA','CHINA', 'EGYPT','GREECE','INDIA','ITALY']
```

مرتب کردن فهرست

مرتب سازی یک فهرست به معنای چیدن عناصر موجود در آن به ترتیبی مشخص است. برای مثال یک فهرست که در بردارنده وزن دانش آموزان یک مدرسه است را می توان به صورت افزایشی (به ترتیب از کمترین وزن به بیشترین وزن) و یا کاهشی (به ترتیب از بیشترین وزن به کم ترین وزن) مرتب کرد. اهمیت مرتب سازی فهرست ها در تاثیر چشمگیر آن بر کاهش زمان جستجو در یک فهرست است الگوریتم های جستجو در یک فهرست مرتب شده نسبت به الگوریتم های جستجو در یک فهرست نامرتب بسیار سریعتر هستند. کلاس `list` در پایتون دارای شگردهای به نام `sort()` است که برای مرتب سازی اشیای از نوع `list` بکار می رود :

مثال :

```
numbers = [17,12,8,20,14,45,78]
```

```
print(numbers)
```

```
#output [17, 12, 8, 20, 14, 45, 78]
```

```
numbers.sort()
```

```
print(numbers)
```

```
#output[8, 12, 14, 17, 20, 45, 78]
```

نوع مرتب سازی این شگرد به صورت پیش فرض افزایشی است و مبنای مرتب سازی آن نیز الفبایی است برای مجبور کردن شگرد `sort()` به مرتب سازی فهرست به صورت کاهشی باید از آرگومان نامدار `reverse` با مقدار `True` استفاده کنید.

مثال :

```
numbers = [17,12,8,20,14,45,78]
```

```
print(numbers) #output [17, 12, 8, 20, 14, 45, 78]
```

```
numbers.sort(reverse = True)
```

```
print(numbers) #output [78, 45, 20, 17, 14, 12, 8]
```

در شگرد `sort()` با استفاده از آرگومان نامدار `key` می توانید نام تابعی که می خواهید عمل مرتب سازی فهرست بر اساس مقدار برگشتی آن انجام شود را به شگرد ارسال کنید برای مثال در کد زیر مرتب سازی فهرست `names` بر اساس خروجی تابع `nameLen` انجام می شود این تابع طول یک رشته را بر می گرداند بنابراین مرتب سازی فهرست `names` بر اساس طول عناصر موجود در فهرست انجام می شود نه بر اساس مقدار آنها.

```
def nameLen(name):
```

```
    return len(name)
```

```
names = ["iran","arash","arman","dariush","ali", "iran manesh"]
```

```
names.sort(key = nameLen )
```

```
print(names)
```

```
#output ['ali', 'iran', 'arash', 'arman', 'dariush', 'iran manesh']
```

شگرد `sort()` حساس به بزرگی و کوچکی حروف است بنابراین از دید این شگرد حروف بزرگ زبان انگلیسی پیش از حروف کوچک قرار می گیرند.

مثال :

```
names = ["arash","Arash","ARash"]
names.sort()
print(names) #output ['ARash', 'Arash', 'arash']
```

برای وادار کردن شگرد `sort()` به مرتب سازی یک فهرست دربردارنده رشته ها بدون توجه به بزرگی و کوچکی حروف می توانید نام تابع `lower()` موجود در کلاس `str` را با استفاده از آرگومان نامدار `key` به این شگرد ارسال کنید.

مثال :

```
names = ["arash","Arash","ARash"]
names.sort(key = str.lower)
print(names) #output ['arash', 'Arash', 'ARash']
```

نسخه برداری^۱ از یک فهرست

فرض کنید فهرستی به نام `list1` دارید و می خواهید از آن نسخه دیگری به نام `list2` داشته باشید به زبان ساده تر قصد دارید تمامی عناصر موجود در فهرست `list1` در فهرست مستقلی به نام `list2` نیز قرارگیرند در نگاه اول به نظر می رسد که ساده ترین راه برآوردن این هدف استفاده از یک دستور انتساب ساده به شکل `list2 = list1` است اما این راه حل نمی تواند منظور ما را برآورده کند چرا که در این حالت هم `list1` و هم `list2` به یک مکان واحد در حافظه اصلی^۲ اشاره خواهند کرد یعنی هر دو دارای نشانی حافظه^۳ یکسان هستند به زبان ساده تر با اجرای دستوری به شکل `list2 = list1` متغیر `list2` تنها یک ارجاع یا اشاره گر به `list1` است و یک فهرست مستقل نیست.

¹ copy

² RAM

³ Memory Address

پس هر تغییری در `list1` به صورت خودکار در `list2` نیز بازتاب خواهد یافت . در پایتون هر متغیری شناسه یکتای خود را دارد که همان نشانی^۱ محل ذخیره سازی آن در حافظه است برای بدست آوردن شناسه یکتای هر متغیر از دستور `id()` استفاده می شود پس بنا برآنچه گفته شد در مثال زیر خروجی دستور `id()` برای هر دو فهرست `list1` و `list2` یکسان خواهد بود :

```
list1 = [۵, ۶, ۸, ۹]
```

```
list2 = list1
```

```
print(id(list1))
```

```
# output ۲۲۹۱۷۵۳۹۴۰۲۸۸
```

```
print(id(list2))
```

```
# output ۲۲۹۱۷۵۳۹۴۰۲۸۸
```

برای چیره شدن بر این مشکل در کلاس `list` شگردي به نام `copy()` پیش بینی شده است که وظیفه آن ایجاد یک فهرست مستقل از یک فهرست موجود است.

مثال :

```
list1 = [10, 15, 20, 18]
```

```
list2 = list1.copy()
```

```
print(list2)
```

روش دیگر نسخه برداری از یک فهرست استفاده از مفهوم انتخاب عناصر فهرست بر اساس محدوده است برای مثال :

```
list1 = [10, 15, 20, 18]
```

```
list2 = list1[:]
```

```
print(list2)
```

¹ Address

پیوند زدن دو یا چندین فهرست به یکدیگر

برای چسباندن دو و یا چند فهرست به یکدیگر چندین روش وجود دارد که عبارتند از :

۱- استفاده از عملگر +

مثال :

```
list1 = [10, 15, 20, 18]
list2 = [7, 5, 9, 17, 22]
list3 = [12, 14, 15]
list4 = list1 + list2 + list3
print(list4)
#outputis : [10, 15, 20, 18, 7, 5, 9, 17, 22, 12, 14, 15]
```

۲- استفاده از حلقه for

مثال :

```
list1 = [10, 15, 20, 18]
list2 = [7, 5, 9, 17, 22]
list3 = [12, 14, 15]
for x in list2:
    list1.append(x)
for x in list3:
    list1.append(x)
print(list1)
```

۳- بکار گیری شگرد extend()

مثال :

```
list1 = [10, 15, 20, 18]
list2 = [7, 5, 9, 17, 22]
list3 = [12, 14, 15]
```

```
list1.extend(list2)
list1.extend(list3)
print(list1)
```

چکیده ای از شگردهای موجود در کلاس list	
شگرد	کارکرد
append()	افزودن یک عنصر به انتهای فهرست
clear()	پاک کردن همه عناصر موجود در یک فهرست
copy()	نسخه برداری از یک فهرست
count()	شمارش تعداد تکرار ورودی در فهرست
extend()	افزودن عناصر موجود در هر نوع قابل پیمایش به انتهای فهرست
index()	بدست آوردن شاخص نخستین عنصر یکسان با ورودی دستور
insert()	افزودن یک عنصر در مکان مشخصی از فهرست
pop()	حذف یک عنصر از مکان مشخصی در فهرست
remove()	حذف نخستین عنصر یکسان با ورودی دستور از فهرست
reverse()	معکوس کردن ترتیب عناصر یک فهرست
sort()	مرتب کردن یک فهرست

فصل دهم

نوع داده tuple

ساختمان داده tuple که در این نوشتار **چندتایی** نامیده می شود یکی دیگر از پرکاربردترین ساختمان داده های موجود در پایتون است و همانند list کاربرد اصلی آن ذخیره چند مقدار مختلف در یک متغیر واحد و دسترسی به همه آنها تنها با یک نام مشخص است. به زبان ساده تر به جای آنکه برای ذخیره موقت هر مقدار، یک متغیر جداگانه تعریف شود، تنها یک متغیر تعریف و تمامی مقادیر مورد نیاز در آن ذخیره و از آن بازیابی می شوند.

در پایتون یک tuple به دو روش زیر ساخته می شود:

۱- مقدار دهی مستقیم به یک متغیر

در این روش مقادیر مورد نظر که با نویسه , از هم جدا می شوند درون یک جفت کمانک باز و بسته () قرار می گیرند و سپس با استفاده از عملگر جایگزینی = در متغیر مورد نظر ذخیره می شوند.

مثال : چندتایی شرکت های فعال در صنعت خودروسازی که دارای چهار مقدار است.

```
cars = ("Toyota", "Peugeot", "Mercedes Benz", "Hundai")
```

```
print(cars)
```

مثال : چندتایی نمرات درس ریاضی دانش آموزان یک کلاس که دارای ۱۰ عنصر است.

```
grades = (18,18.30,12,17.5,16,19,17,14.5,13,15)
```

```
print(grades)
```

برای ساخت یک چندتایی تک عنصری یعنی چندتایی که تنها یک مقدار را در خود نگه می دارد باید پس از نخستین عنصر آن یک نویسه , نوشته شود در غیر این صورت پایتون نمی تواند تشخیص دهد که هدف شما ساخت یک tuple است :

مثال :

```
thistuple = ("apple",)
print(type(thistuple)) # output <class 'tuple'>

thistuple = ("apple")
print(type(thistuple)) # output <class 'str'>
```

۲- استفاده از سازنده^۱ کلاس tuple

مثال : چندتایی نام دانش آموزان یک کلاس

```
student = tuple(("Arash" , "Arman" , "Mina" , "Darush" , "Sepideh"))
print(student)
```

مثال : چندتایی نمرات درس ریاضی تعدادی از دانشجویان

```
grades = tuple((18,18.30,12,17.5,16,19,17,14.5,13,15))
print(grades)
```

چندتایی ها دارای ویژگی های زیر هستند:

۱- عناصر درون یک tuple دارای ترتیب^۲ و نظم خاصی هستند نه به این معنی که عناصر از کوچکترین مقدار به بزرگترین مقدار و یا برعکس مرتب شده اند بلکه به این معنی که عناصر درون آن دارای ترتیب قرار گیری مشخصی هستند به زبان ساده تر محل قرار گرفتن عناصر یک چندتایی بر اساس ترتیب درج آنها در tuple و یا به هنگام تعریف و مقدار دهی اولیه به آن تعیین می شود

¹ Constructor

² Ordered

به عددی که نشان دهنده محل قرارگرفتن هر عنصر درون چندتایی است شاخص^۱ گفته می شود. شاخص گذاری عناصر موجود در یک tuple از عدد صفر شروع می شود یعنی نخستین عنصر دارای شاخص صفر، دومین عنصر دارای شاخص یک و به همین ترتیب آخرین عنصر دارای شاخص n-1 است که n طول چندتایی یا همان تعداد عناصر موجود در چندتایی است. ناگفته نماند که کلاس tuple دارای شگردهایی^۲ است که می توانند ترتیب قرار گرفتن عناصر در یک چندتایی را برهم بزنند اما در حالت کلی یک چندتایی دارای ویژگی ترتیب است.

۲- چندتایی یک شی غیر قابل تغییر^۳ است به این معنی که پس از تعریف و مقدار دهی اولیه به آن نمی توان عناصر موجود در آن را حذف کرد، تغییر داد و یا عناصری جدیدی به آن افزود.

۳- عناصر درون یک چندتایی می توانند تکراری باشند. توانایی چندتایی ها در پذیرش مقادیر تکراری مرهون شاخص گذاری عناصر موجود در آن است، تا زمانی که هر دو مقدار یکسان شاخص متفاوتی داشته باشند هیچ مشکلی پیش نخواهد آمد.

مثال: چندتایی زیر دارای دو عنصر با مقدار تکراری "Blue" و شاخص های متفاوت ۰ و ۴ است

```
colorList = ("Blue", "Red", "Purple", "Green", "Blue")
print(colorList)
```

۴- هر یک از عناصر موجود در یک چندتایی می توانند نوع داده متفاوتی داشته باشند

مثال: در چندتایی myFavoritWords نخستین عنصر از نوع رشته، دومین عنصر یک عدد صحیح و سومین عنصر یک عدد اعشاری است.

```
myFavoritWords = ("Hundai", ۲۰, ۵۶/۵)
print(myFavoritWords)
```

¹ index

² Methods

³ unchangeable

روش دستیابی به عناصر یک چندتایی

همانگونه که گفته شد عناصر موجود در چندتایی شاخص گذاری می شوند یعنی محل قرارگیری عناصر موجود در آن به ترتیب و با شروع از عدد صفر شماره گذاری می شود بنا براین می توان با قراردادن شاخص هر عنصر درون عملگر انتخاب (جفت علامت []) که پس از نام چندتایی می آید به مقدار آن دست یافت برای نمونه درمثال زیر `student[0]` نخستین عنصر چندتایی یعنی "Arash" و `student[3]` چهارمین عنصر چندتایی یعنی "Bahram" را بازیابی می کند.

مثال :

```
student = ("Arash" , "Arman" , "Mina" , "Bahram" , "Sepideh")
print(student[0])
print(student[1])
print(student[2])
print(student[3])
print(student[4])
```

چندتایی ها از شاخص گذاری منفی هم پشتیبانی می کنند بدین معنی که آخرین عنصر یک چندتایی یعنی نخستین عنصر از انتها دارای شاخص 1- دومین عنصر از انتهای شاخص 2- و به همین ترتیب تا آخرین عنصر از انتهای چندتایی که دارای شاخص n- است که n تعداد کل عناصر موجود در چندتایی است.

مثال:

```
student = ("Arash" , "Arman" , "Mina" , "Bahram" , "Sepideh")
```

شاخص گذاری منفی	-۵	-۴	-۳	-۲	-۱
شاخص گذاری مثبت	0	1	2	3	4
Student List	Arash	Arman	Mina	Bahram	Sepideh

مثال: دستور `print(thistuple[-1])` در کد زیر عنصر "Third" را چاپ خواهد کرد.

```
thistuple = ("First", "Second", "Third")
```

```
print(thistuple[-1])
```

دیگر روش دستیابی به عناصر موجود در یک چندتایی استفاده از حلقه `for` به همراه عملگر `in` است.

مثال :

```
student = ("Arash", "Arman", "Mina", "Darush", "Sepideh")
```

```
for itm in student:
```

```
    print(itm)
```

طول چندتایی

به تعداد عناصر موجود در یک چندتایی طول آن گفته می شود برای بدست آوردن طول یک چندتایی می توانید از تابع `len()` استفاده کنید.

مثال :

```
grades = (18,18.30,12,17.5,16,19,17,14.5,13,15)
```

```
length = len(grades)
```

```
print(length) #OutPut is 10
```

مثال : کد زیر عناصر موجود در چندتایی `student` را از انتها به ابتدا چاپ می کند. برای درک بهتر کد دوباره یادآوری می شود که چندتایی ها از شاخص گذاری منفی عناصر پشتیبانی می کنند بدین معنی که آخرین عنصر دارای شاخص 1- دومین عنصر از انتهای شاخص 2- و به همین ترتیب تا آخرین عنصر از انتهای چندتایی که دارای شاخص -n است که n تعداد کل عناصر موجود در چندتایی است.


```
student = tuple(("Arash" , "Arman", "Mina" ,"Bahram", "Sepideh"))  
end = -1  
start = -len(student)  
while end >= start:  
    print(student[end])  
    end += -1
```

انتخاب محدوده معینی از یک چندتایی

می توان محدوده ای از عناصر موجود در یک چندتایی را انتخاب کرد برای این کار باید شاخص عناصر موجود در محدوده مورد نظر را به صورت Start:End درون عملگر انتخاب ([]) قرار دهید با مشخص کردن محدوده مورد نظر به صورت [Start:End] عناصری که دارای شاخص Start تا End-1 هستند انتخاب می شوند. به یاد داشته باشید که شاخص End را یک واحد بیش از شاخص آخرین عنصر مورد نظر انتخاب کنید. مقدار بیش فرض شاخص Start صفر است پس اگر مقدار آن خالی رها شود محدوده انتخاب از نخستین عنصر موجود در چندتایی شروع می شود همچنین مقدار بیش فرض شاخص End برابر با طول چندتایی است پس چنانچه مقدار آن خالی رها شود محدوده انتخاب از شاخص Start آغاز و تا آخرین عنصر موجود در چندتایی که دارای شاخص n-1 است ادامه خواهد یافت. (n طول چندتایی است)

مثال:

```
thistuple = ("Arash" , "Arman", "Mina" ,"Bahram", "Sepideh")  
print(thistuple [2:4])
```

خروجی کد بالا "Mina", "Bahram" است

مثال:

```
tuple1 = (10,20,30,40,50)
tuple2 = tuple1[:]
print(tuple2)
#output is : (10, 20, 30, 40, 50)
```

کد بالا تمامی عناصر موجود در tuple1 را انتخاب و در tuple2 قرار می دهد

مثال:

```
thistuple = ("Arash" , "Arman" , "Mina" ,"Bahram" , "Sepideh")
partuple = thistuple [:4]
print(partuple)
```

کد بالا عناصر "Arash" , "Arman" , "Mina" ,"Bahram" را چاپ می کند

مثال :

```
thistuple = ("Arash" , "Arman" , "Mina" ,"Bahram" , "Sepideh")
print(thistuple[2:])
```

مثال : کد زیر عناصر "Mina" ,"Bahram" را چاپ می کند

```
thistuple = ("Arash" , "Arman" , "Mina" ,"Bahram" , "Sepideh")
part1 = thistuple [-3:-1]
print(part1)
```

برای اطمینان از موجود بودن عنصر مورد نظر در یک چندتایی مشخص می توانید از دستور if به همراه عملگر in استفاده کنید

مثال:

```
thistuple = ("Arash" , "Arman" , "Mina" ,"Bahram" , "Sepideh")
if "Arman" in thistuple:
    print("Arman is in the tuple")
```

برای اطمینان از اینکه عنصر مورد نظر در چندتایی وجود ندارد می توانید از دستور if به همراه عملگر not in استفاده کنید

مثال :

```
thistuple = ["Arash" , "Arman" , "Mina" ,"Bahram" , "Sepideh"]
if "Amir" not in thistuple:
    print("Amir is not in the tuple")
```

بروزرسانی مقدار یک عنصر در tuple

همانگونه که گفته شد چندتایی یک شی غیرقابل تغییر است به این معنی که پس از تعریف و مقدار دهی اولیه به tuple نمی توان عناصر موجود در آن را حذف کرد ، تغییر داد و یا عناصری جدیدی به آن افزود. از این رو تنها راه باقی مانده برای به روزرسانی مقادیر موجود در آن این است که ابتدا tuple به یک list تبدیل شود ، تغییرات مورد نظر در list داده شود و سرانجام list ویرایش شده دوباره به tuple تبدیل گردد :

مثال:

```
averages = (19, 18, 16 , 18.8, 17, 15.5, 19.8 , 20 , 14)
list2 = list(averages)
list2[0] = 19.7
list2[2] = 17
averages = tuple(list2)
print(averages)
```

افزودن یک عنصر به tuple

چندتایی یک ساختمان داده تغییرناپذیر است از این رو همانند list شگردی به نام append() برای آن پیش بینی نشده است بنابراین برای افزودن یک مقدار جدید به آن دو راه وجود دارد :

۱- تبدیل tuple به list، افزودن مقدار مورد نظر به list و تبدیل دوباره آن به tuple

مثال:

```
averages = (19, 18, 16)
list2 = list(averages)
list2.append(25)
averages = tuple(list2)
print(averages)
```

۲- کلاس tuple از عملگر الحاق (+) پشتیبانی می کند بنابراین برای افزودن یک عنصر به یک tuple می توانید یک چندتایی تک عنصری بسازید و آنرا به tuple مورد نظر الحاق کنید.

مثال :

```
tpl1 = (5,6,7)
tpl2 = (9,)
tpl1 += tpl2
print(tpl1)
```

حذف یک عنصر از tuple

با توجه به تغییرناپذیر بودن چندتایی ها ، تنها راه باقی مانده برای حذف یک عنصر از آن این است که ابتدا tuple به یک list تبدیل شود ، عنصر مورد نظر از list حذف شود و سرانجام list ویرایش شده دوباره به tuple تبدیل گردد :

مثال :

```
tpl1 = (5,6,7,10)
list1 = list(tpl1)
list1.remove(10)
tpl1 = tuple(list1)
print(tpl1)
```

برای حذف تمام یک tuple می توانید از دستور del استفاده کنید :

مثال :

```
tpl1 = (5,6,7,10)
del(tpl1)
print(tpl1)
```

در کد بالا چندتایی tpl1 به وسیله دستور del حذف می شود بنابراین با رسیدن اجرا به دستور print(tpl1) خطای زیر توسط مفسر پایتون اعلام خواهد شد :

NameError: name 'tpl1' is not defined

بسته بندی و بازپخش چندتایی ها

به ساخت یک tuple از راه تعیین یکایک عناصر آن بسته بندی^۱ tuple گفته می شود

مثال:

```
tuple1 = (5, 6, 7, 10, 12)
```

در پایتون با استفاده از عملگر جایگزینی = می توانید عناصر موجود در یک tuple را به دسته ای از متغیرها اختصاص دهید. که به این کار توزیع یا بازپخش و یا بازکردن^۲ بسته بندی یک tuple گفته می شود.

مثال:

```
tuple1 = (5, 6, 7, 10)
a, b, c, d = tuple1
print(a)
print(b)
print(c)
print(d)
```

¹ packing

² unpacking

مثال:

```
fruits = ("apple", "banana", "cherry")
(green, yellow, red) = fruits
print(green)
print(yellow)
print(red)
```

به هنگام بازپخش یک tuple تعداد متغیرها باید برابر با تعداد عناصر موجود در چندتایی باشند. در غیر این صورت مفسر پایتون خطا اعلام خواهد کرد. برای پیشگیری از این وضعیت می توانید پیش از نام یکی از متغیرها علامت * قرار دهید در این حالت دیگر عناصر موجود در tuple که متغیری متناظر با آنها وجود ندارد در قالب یک list در آن قرار خواهند گرفت.

مثال :

```
fruits = ("apple", "banana", "cherry", "orange", "tomato")
(green, yellow, *red) = fruits
print(green)
print(yellow)
print(red) # output ['cherry', 'orange', 'tomato']
```

به یاد داشته باشید که اختصاص مقادیر موجود در tuple به متغیرها بر اساس ترتیب آنها و با رعایت برابری تعداد متغیرها و تعداد عناصر tuple است بنابراین اگر علامت * در کنار متغیری به جز آخرین متغیر قرار گیرد رفتار بازپختی از این اصول پیروی خواهد کرد

مثال: در کد زیر علامت * در کنار دومین متغیر قرار گرفته است

```
fruts = ("apple", "banana", "cherry", "orange", "tomato")
(green, *yellow, red) = fruts
print(green)
print(yellow)
```

```
print(red)
```

```
# output is :
```

```
# apple
```

```
# ['banana', 'cherry', 'orange']
```

```
# tomato
```

مثال: در کد زیر علامت * در کنار نخستین متغیر قرار گرفته است

```
fruits = ("apple", "banana", "cherry", "orange", "tomato")
```

```
(*green, yellow, red) = fruits
```

```
print(green)
```

```
print(yellow)
```

```
print(red)
```

```
#output :
```

```
#['apple', 'banana', 'cherry']
```

```
#orange
```

```
#tomato
```

پیمایش چندتایی ها

پیمایش یک چندتایی به معنای حرکت در میان عناصر موجود در آن و خواندن مقدار هر یک از آنها به ترتیب ملاقات است. پیمایش می تواند از یک نقطه دلخواه آغاز و در یک نقطه دلخواه دیگر به پایان برسد. در بسیاری از کاربردها نخستین عنصر چندتایی نقطه آغاز پیمایش است و آخرین عنصر موجود در چندتایی نقطه پایان، جهت پیمایش می تواند از ابتدا به انتهای چندتایی و یا برعکس باشد. پیمایش یک چندتایی با استفاده از حلقه ها انجام می شود:

مثال: پیمایش چندتایی Customers با استفاده از حلقه for

```
Customers = ("arash", "Arman", "Farhad", "Kaveh")
```

```
for itm in Customers:
```

```
    print(itm)
```

مثال: پیمایش چندتایی Customers با استفاده از حلقه while

```
Customers = ("arash" , "Arman", "Farhad", "Kaveh")
i=0
while i < len(Customers):
    print(Customers[i])
    i += 1
```

مثال: پیمایش چندتایی Customers با بهره‌گیری از حلقه for ، مفهوم شاخص و دستور range()

```
Customers = ("arash" , "Arman", "Farhad", "Kaveh")
for i in range(len(Customers)):
    print(Customers[i])
```

الحاق یا به هم پیوستن چندتایی ها

برای به هم پیوستن دو یا چند tuple به یکدیگر می‌توانید از عملگر الحاق (+) استفاده کنید

مثال:

```
tuple1 = (10, 15, 20, 18)
tuple2 = (7, 5, 9, 17, 22)
tuple3 = (12, 14, 15)
tuple4 = tuple1 + tuple2 + tuple3
print(tuple4)
#outputis : (10, 15, 20, 18, 7, 5, 9, 17, 22, 12, 14, 15)
```

می‌توان یک tuple را در یک عدد ضرب کرد نتیجه یک tuple است که اعضای آن به تعداد عدد ضرب شده تکرار شده‌اند.

مثال:

```
tuple1 = (5,6,7)
tuple2 = tuple1 * 3
```



```
print(tuple2) #outputis : (5, 6, 7, 5, 6, 7, 5, 6, 7)
```

شگرد های پر کاربرد در چندتایی ها

شگرد count()

مقداری به عنوان ورودی می پذیرد و تعداد تکرار آن در tuple را برمی گرداند اگر مقدار ارسال شده به شگرد در چندتایی مورد نظر وجود نداشته باشد مقدار برگشتی عدد صفر خواهد بود.

مثال:

```
tuple1 = (5,6,7,6,7,8,6,5,6)
cnt = tuple1.count(6)
print(cnt)
#output is : 4
```

شگرد index()

مقداری به عنوان ورودی می پذیرد و نخستین محل قرارگیری آن در tuple یا همان شاخص عنصر را برمی گرداند. به یاد داشته باشید که شاخص گذاری عناصر یک tuple از عدد صفر شروع می شود. اگر مقدار ارسال شده به شگرد در چندتایی مورد نظر وجود نداشته باشد مفسر پایتون خطای ValueError: tuple.index(x): x not in tuple را صادر خواهد کرد.

مثال:

```
tuple1 = (5,6,7,6,7,8,6,5,6)
pos = tuple1.index(6)
print(pos)
#outputis : 1
```

تابع zip()

گاهی نیاز است که عناصر نظیر به نظیر موجود در دو یا چند شی قابل پیمایش (set, tuple, list) را در قالب یک tuple ترکیب و تمامی چندتایی های بدست آمده را در یک شی قابل پیمایش دیگر ذخیره کنیم برای مثال فرض کنید list1 در بردارنده نام دانش آموزان و list2 در بردارنده

معدل همان دانش آموزان به ترتیب نام آنها در `list1` باشد. می خواهیم فهرستی از `tuple` های در بردارنده نام و معدل هردانش آموز را در `list3` ذخیره کنیم به این منظور می توان کدی همانند زیر نوشت :

```
list1 = ['arash', 'arman', 'sepideh', 'dariush']
list2 = [19.5, 19, 18, 17.5]
list3 = []
i = 0
for x in list1:
    name,avrg = x,list2[i]
    t = tuple((name,avrg))
    list3.append(t)
    i += 1
print(list3)
```

خروجی کد بالا همانند زیر است:

```
[('arash', 19.5), ('arman', 19), ('sepideh', 18), ('dariush', 17.5)]
```

راه حل ساده تری هم برای انجام این کار وجود دارد و آن بکارگیری تابع `zip()` است ، این تابع چندین شی مختلف از انواع داده قابل پیمایش^۱ مانند `list` ، `tuple` و `set` را به عنوان ورودی می پذیرد و یک شی واحد قابل پیمایش از نوع `zip` برمی گرداند که دربردارنده تعدادی `tuple` است هر `tuple` حاوی عناصر متناظر موجود در هر یک از اشیای قابل پیمایش ورودی به تابع `zip()` است . برای نمونه مثال بالا با استفاده از این تابع به شکل زیر برنامه نویسی می شود :

```
list1 = ['arash', 'arman', 'sepideh', 'dariush']
list2 = [19.5, 19, 18, 17.5]
list3 = list(zip(list1, list2))
print(list3)
```

خروجی کد بالا همانند زیر است :

```
[('arash', 19.5), ('arman', 19), ('sepideh', 18), ('dariush', 17.5)]
```

¹Iterable

مثال: فرض کنید سه فهرست به نام های raining99، raining400 و raining401 که به ترتیب دربردارنده میزان فرضی بارش باران در ۱۲ ماه سالهای ۱۳۹۹، ۱۴۰۰ و ۱۴۰۱ در استان خراسان شمالی هستند را در اختیار دارید و می خواهید جمع ماهانه بارش باران در سه سال یاد شده را محاسبه و در فهرست چهارمی به نام totalRaining وارد کنید کد نویسی مسئله بدون استفاده از تابع zip() همانند زیر است :

```
raining99 = [66.1,67,67.2,60,59,50,49,48,50,47.3,38.3,37]
raining400 = [65.1,66,68.2,57,55,51,47,48,46,46.3,39.1,36]
raining401 = [66.5,68,66.7,61.1,56,44.2,44,45,47,47.3,38.3,33]
totalRaining = []
i = 0
while i < 12:
    totalRaining.append(raining99[i] + raining400[i] + raining401[i])
    i += 1
print(totalRaining)
```

کد نویسی مسئله با استفاده از تابع zip() همانند زیر خواهد بود:

```
raining99 = [66.1,67,67.2,60,59,50,49,48,50,47.3,38.3,37]
raining400 = [65.1,66,68.2,57,55,51,47,48,46,46.3,39.1,36]
raining401 = [66.5,68,66.7,61.1,56,44.2,44,45,47,47.3,38.3,33]
temp = list(zip(raining99, raining400, raining401))
totalRaining = [sum(x) for x in temp]
print(totalRaining)
```

فصل یازدهم

نوع داده dictionary

ساختمان داده dictionary که در این نوشتار واژه نامه خوانده می شود یکی دیگر از پرکاربردترین ساختمان داده های موجود در پایتون است و همانند list و tuple کاربرد اصلی آن ذخیره چندین مقدار مختلف در یک متغیر واحد و دسترسی به همه آنها تنها با یک نام مشخص است. واژه نامه داده ها را به صورت زوج های مقدار: کلید (key:value) ذخیره می کند. هر جفت key:value مقداری را به یک کلید یکتا نگاشت می کند. به زبان ساده تر در واژه نامه هر مقدار دارای کلید ویژه خود است. هر کلید در تمامی dictionary یکتاست اما مقدار ها می توانند تکراری باشند.

در پایتون یک dictionary به دو روش زیر ساخته می شود:

۱- مقدار دهی مستقیم یک متغیر

در این روش زوج های key:value که با نویسه , از هم جدا می شوند درون یک جفت علامت کمان باز و بسته ({}) قرار می گیرند و سپس با استفاده از عملگر جایگزینی = در متغیر مورد نظر ذخیره می شوند.

مثال: واژه نامه فرضی شماره تماس نمایندگی شرکت های فعال در صنعت خودروسازی در ایران

```
carsDict = {'Toyota': '021555', 'Peugeot': '021888', 'Hundai': '021666'}  
print(carsDict)
```

مثال: واژه نامه در بردارنده اطلاعات یک کارمند شامل : نام ، نام خانوادگی ، حقوق ، وزن و قد

```
employees = {  
    'name': 'arash',  
    'family': 'izanlou',  
    'salary': 70000000,  
    'height': 1.85,  
    'weight': 78}  
print(employees)
```

۲- استفاده از سازنده کلاس **dictionary**

مثال:

```
student = dict(name = "arash" , Average = 19.5, schoolName =  
"Farhangian")  
print(student)
```

مثال: ساخت واژه نامه از روی یک **list** در بردارنده **tuple** هایی که هر **tuple** بیانگر ارتفاع از سطح دریا های آزاد یک پایتخت است

```
list1 = [('tehran', 1830), ('paris', 34), ('Tokyo', 17)]  
myDict = dict(list1)  
print(myDict)
```

مثال:

```
a = dict(one=1, two=2, three=3)  
b = {'one': 1, 'two': 2, 'three': 3}  
c = dict(zip(['one', 'two', 'three'], [1, 2, 3]))
```

```
d = dict([('two', 2), ('one', 1), ('three', 3)])
e = dict({'three': 3, 'one': 1, 'two': 2})
f = dict({'one': 1, 'three': 3}, two=2)
print(a == b == c == d == e == f) #output is : True
print(a) #output is: {'one': 1, 'two': 2, 'three': 3}
```

۳- استفاده از شیوه واژه نامه برگزیده

واژه نامه برگزیده یک شیوه ساده در پایتون برای ساخت یک واژه نامه جدید از روی یک شی قابل پیمایش موجود است. برای مثال می خواهیم واژه نامه ای ایجاد کنیم که در هر زوج key:value آن اعداد ۱ تا ۶ به عنوان کلید و مربع هر عدد به عنوان مقدار در نظر گرفته شوند:

```
myDict = {x:x**۲ for x in range(1,7)}
print(myDict)
#output : {۱: ۱, ۲: ۴, ۳: ۹, ۴: ۱۶, ۵: ۲۵, ۶: ۳۶}
```

مثال :

```
keys = {100, 102, 200, 202}
values = {'arash', 'arman', 'sepideh', 'dariush'}
mydict = { k : v for (k, v) in zip(keys, values) }
print(mydict)
```

مثال :

```
dict1 = { x.upper() : x * 3 for x in "iran" }
print(dict1)
#output is : {'I': 'iii', 'R': 'rrr', 'A': 'aaa', 'N': 'nnn'}
```

ویژگی های واژه نامه

۱- واژه نامه یک شی قابل تغییر^۱ است به این معنی که پس از تعریف و مقدار دهی اولیه به آن می توان عناصر موجود در آن را حذف کرد ، تغییر داد و یا عناصری جدیدی به آن افزود.

۲- در زوج های `key:value` ذخیره شده در یک واژه نامه کلید یکتاست و نمی تواند تکراری باشد اما مقادارها می توانند تکراری باشند. به زبان ساده تر هیچ دو مقداری در یک واژه نامه دارای کلید یکسان نیستند.

۳- در زوج های `key:value` موجود در یک واژه نامه ، مقدار (`value`) می تواند هر نوع داده ای را داشته باشد یعنی می تواند از نوع عددی ، رشته ، `list` ، `tuple` و ... باشد.

روش دستیابی به عناصر یک واژه نامه

عناصر موجود در یک واژه نامه یا همان زوج های `key:value` با استفاده از کلید خود ، شاخص گذاری می شوند بنا براین می توان با قراردادن کلید هر عنصر درون عملگر انتخاب (جفت علامت `[]`) که پس از نام واژه نامه می آید) به مقدار آن دست یافت.

مثال :

```
student = dict(name = "arash" , Average = ۱۹/۵, schoolName = "Farhangian")
```

```
print(student['name'])
```

```
print(student['Average'])
```

```
print(student['schoolName'])
```

روش دیگر دستیابی به مقدار ذخیره شده در هر یک از زوجهای `key:value` استفاده از شگردهای `get()` موجود در کلاس `dictionary` است.

¹ Changeble

مثال :

```
student = dict(name = "arash" , Average = 19.5, schoolName = "Farhangian")
```

```
print(student.get('name'))
```

```
print(student.get('Average'))
```

```
print(student.get('schoolName'))
```

دیگر روش دستیابی به عناصر موجود در یک واژه نامه ، بکارگیری شگرد `items()` است. این شگرد هر زوج `key:value` موجود در واژه نامه را در قالب یک `tuple` به شکل `(key, value)` ذخیره می کند سپس تمامی `tuple` های بدست آمده را در یک شی قابل پیمایش از نوع `dict_items` قرار می دهد. مهم است بدانید که شی برگشت داده شده توسط این شگرد یک نما (`view`) از واژه نامه است بنابراین هر تغییری در واژه نامه بی درنگ در آن بازتاب خواهد یافت.

مثال :

```
student = dict(name = "arash" , Average = 19.5, schoolName = "Farhangian")
```

```
studentItems= student.items()
```

```
print(studentItems)
```

```
#output dict_items([('name', 'arash'), ('Average', 19.5), ('schoolName', 'Farhangian')])
```

شی بدست آمده از اجرای شگرد `items()` قابل پیمایش است بنابراین می توان همانند مثال زیر آن را پیمایش کرد :

مثال :

```
student = dict(name = "arash" , Average = 19.5, schoolName =  
"Farhangian")  
  
for itm in student.items():  
  
    print(itm)
```

خروجی کد بالا همانند زیر است :

```
('name', 'arash')  
('Average', 19.5)  
('schoolName', 'Farhangian')
```

بدست آوردن کلید های یک dictionary در قالب یک شی قابل پیمایش

شگرد `keys()` موجود در کلاس `dictionary` فهرستی از کلیدهای موجود در واژه نامه را در قالب یک شی قابل پیمایش از نوع `dict_keys` برمی گرداند. شی برگشت داده شده توسط این شگرد یک نما (`view`) از کلید های واژه نامه است بنابراین هر تغییری در کلیدهای واژه نامه بی درنگ در آن بازتاب خواهد یافت.

مثال :

```
student = dict(name = "arash" , Average = 19.5, schoolName =  
"Farhangian")  
  
studentKeys = student.keys()  
  
for k in studentKeys:  
  
    print(k)
```

بدست آوردن مقدارهای یک dictionary در قالب یک شی قابل پیمایش

شگرد `values()` موجود در کلاس `dictionary` فهرستی از مقدارهای موجود در واژه نامه را در قالب یک شی قابل پیمایش از نوع `dict_values` برمی گرداند. شی برگشت داده شده توسط این شگرد یک نما (`view`) از مقدارهای ذخیره شده در واژه نامه است بنابراین هر تغییری در مقدارهای واژه نامه بی درنگ در آن بازتاب خواهد یافت.

مثال :

```
student = dict(name = "arash" , Average = 19.5, schoolName =
"Farhangian")
studentValues = student.values()
print(type(studentValues))
for v in studentValues:
    print(v)
```

طول واژه نامه

به تعداد زوج های `key:value` موجود در یک واژه نامه طول واژه نامه گفته می شود برای بدست آوردن طول یک واژه نامه می توانید از تابع `len()` استفاده کنید.

مثال :

```
student = dict(name = "arash" , Average = 19.5, schoolName =
"Farhangian")
length = len(student)
print(length) #OutPut is 3
```

برای اطمینان از موجود بودن یک کلید در واژه نامه مورد نظر می توانید از دستور `if` به همراه عملگر `in` استفاده کنید

مثال :

```
student = dict(name = "arash" , Average = 19.5, schoolName =  
"Farhangian")  
if "name" in student:  
    print("name key is in student dictionary")
```

یا

```
student = dict(name = "arash" , Average = ۱۹/۵, schoolName =  
"Farhangian")  
if "name" in student.keys():  
    print("name key is in student dictionary")
```

برای اطمینان از اینکه کلید مورد نظر در واژه نامه وجود ندارد می توانید از دستور `if` به همراه عملگر `not in` استفاده کنید

مثال :

```
student = dict(name = "arash" , Average = 19.5, schoolName =  
"Farhangian")  
if "family" not in student:  
    print("family key is not in student dictionary")
```

یا

```
student = dict(name = "arash" , Average = 19.5, schoolName =  
"Farhangian")  
if "family" not in student.keys():  
    print("family key is not in student dictionary")
```

به روزرسانی مقدار یک عنصر در dictionary

همانگونه که گفته شد واژه نامه یک شی قابل تغییر است به این معنی که پس از تعریف و مقدار دهی اولیه به آن می توان عناصر موجود در آن را حذف کرد ، تغییر داد و یا عناصری جدیدی به آن افزود. برای به روزرسانی مقدار یک کلید در dictionary دو روش وجود دارد :

۱- دستیابی به مقدار یک کلید و تغییر آن با استفاده از عملگر انتخاب [] و سپس انتساب = به شکل زیر:

```
yourDic['key '] = newValue
```

مثال:

```
CarDict = {'brand':'Toyota',
           'model':'prado',
           'year':2009}
CarDict['year '] = 2022
print(CarDict)
```

۲- استفاده از شگرد update() موجود در کلاس dictionary

شگرد update() می تواند یک ورودی از نوع dictionary و یا هر شی قابل پیمایش دیگری که در بردارنده داده به شکل زوج های key:value باشد را پذیرفته و مقدار کلیدهای متناظر با ورودی خود را در واژه نامه تغییر دهد:

مثال :

```
CarDict = {'brand':'Toyota',
           'model':'prado',
           'year':2009}

CarDict.update({'brand':'Ford', 'model':'Mustang', 'year':1964})
print(CarDict)
#output {'brand': 'Ford', 'model': 'Mustang', 'year': 1964}
```

افزودن یک عنصر به dictionary

واژه نامه یک ساختمان داده تغییر پذیر است از این رو می توان عنصر جدیدی به آن افزود برای این منظور می توان عملگر انتخاب [] که دربردارنده کلید جدید باشد را همانند زیر و با استفاده از عملگر انتساب = مقداردهی کرد. با این کار در واژه نامه مورد نظر زوج key:value جدیدی ساخته می شود.

```
yourdict['NewKey'] = Value
```

مثال :

```
CarDict = {'brand':'Toyota',  
           'model':'prado',  
           'year':2009}
```

```
CarDict['country'] = 'japan'
```

```
print(CarDict)
```

```
#output {'brand': 'Toyota', 'model': 'prado', 'year': 2009, 'country':  
'japan'}
```

شیوه دیگر افزودن یک عنصر به واژه نامه بکارگیری شگرد update() است که در بخش پیشین با آن آشنا شدید. این شگرد رفتار جالبی در رفتار با زوج های کلید : مقدار موجود در آرگومان دریافتی خود دارد بدین معنی که اگر هر یک از زوج های key:value موجود در آرگومان ، متناظر با هیچ یک از زوج های key:value موجود در واژه نامه نباشد یعنی کلید آن با هیچ یک از کلیدهای واژه نامه برابر نباشد زوج key:value به صورت یک عنصر جدید به واژه نامه افزوده خواهد شد

مثال :

```
CarDict = {'brand':'Toyota',
           'model':'prado',
           'year':2009}
CarDict.update({'country':'japan'})
print(CarDict)
#output {'brand': 'Toyota', 'model': 'prado', 'year': 2009, 'country':
'japan'}
```

حذف یک عنصر از dictionary

برای حذف یک عنصر از واژه نامه چند روش مختلف وجود دارد که در زیر به برخی از این روشها می پردازیم :

۱- شگرد `pop()` یک عنصر دارای کلید مشخص را از واژه نامه حذف می کند

مثال :

```
CarDict = {'brand':'Toyota',
           'model':'prado',
           'year':2009}

CarDict.pop('year')
print(CarDict)
#output {'brand': 'Toyota', 'model': 'prado'}
```

۲- تابع `del()` می تواند یک عنصر دارای کلید مشخص را از واژه نامه حذف کند.

مثال :

```
CarDict = {'brand':'Toyota',  
           'model':'prado',  
           'year':2009}  
  
del(CarDict['year'])  
print(CarDict)  
#output {'brand': 'Toyota', 'model': 'prado'}
```

۳- برای حذف کل یک dictionary می توانید از تابع `del()` استفاده کنید :

مثال :

```
CarDict = {'brand':'Toyota',  
           'model':'prado',  
           'year':2009}  
  
del(CarDict)  
print(CarDict)
```

در کد بالا واژه نامه `CarDict` به وسیله دستور `del` حذف می شود بنابراین با رسیدن اجرا به دستور `print(CarDict)` خطای زیر توسط مفسر پایتون اعلام خواهد شد :

`NameError: name 'CarDict' is not defined`

۴- شگرد `clear()` برای پاک کردن تمامی زوجهای `key:value` موجود در یک واژه نامه استفاده می شود این شگرد تمامی عناصر موجود در واژه نامه را پاک می کند اما خود واژه نامه را باقی می گذارد و دوباره می توان عناصر جدیدی در آن ذخیره کرد.

مثال:

```
CarDict = {'brand':'Toyota',
           'model':'prado',
           'year':2009}
```

```
CarDict.clear()
print(CarDict)
#output {}
```

پیمایش واژه نامه

پیمایش یک واژه نامه به معنای حرکت در میان عناصر آن و خواندن مقدار هر یک از آنها به ترتیب ملاقات است. پیمایش یک فهرست با استفاده از حلقه for انجام می شود در این حالت آنچه که برگردانده می شود کلیدهای موجود در dictionary است نه مقادیر موجود در آن :

مثال :

```
CarDict = {'brand':'Toyota',
           'model':'prado',
           'year':2009}
for x in CarDict:
    print(x)
```

خروجی کد بالا به شکل زیر است :

```
brand
model
year
```

مثال : چاپ تمامی مقادیر موجود در واژه نامه CarDict به روش پیمایش

```
CarDict = {'brand':'Toyota',
           'model':'prado',
           'year':2009}
for x in CarDict:
```



```
print(CarDict[x])
```

خروجی کد بالا به شکل زیر است :

```
Toyota  
prado  
2009
```

برای پیمایش یک واژه نامه می توانید از شگرد `keys()` در حلقه `for` هم استفاده کنید :

مثال :

```
CarDict = {'brand':'Toyota',  
           'model':'prado',  
           'year':2009}  
for k in CarDict.keys():  
    print(k)
```

برای پیمایش تمامی مقادیر موجود در یک واژه نامه می توانید از شگرد `values()` در حلقه `for` استفاده کنید :

مثال :

```
CarDict = {'brand':'Toyota',  
           'model':'prado',  
           'year':۲۰۰۹}  
for v in CarDict.values():  
    print(v)
```

برای پیمایش همزمان کلیدها و مقادیر می توان از شگرد `items()` بهره برد :

مثال :

```
CarDict = {'brand':'Toyota',
           'model':'prado',
           'year':2009}
for k, v in CarDict.items():
    print(k, "--- ", v)
```

نسخه برداری از یک واژه نامه

فرض کنید واژه نامه‌ای به نام dict1 دارید و می‌خواهید از آن نسخه‌ای به نام dict2 داشته باشید به زبان ساده‌تر قصد دارید تمامی عناصر موجود در واژه‌نامه dict1 در واژه‌نامه مستقلی به نام dict2 نیز قرارگیرند در نگاه اول به نظر می‌رسد که ساده‌ترین راه برآوردن این هدف استفاده از دستور انتساب ساده dict2 = dict1 باشد اما این راه حل نمی‌تواند منظور ما را برآورده کند چرا که در این حالت dict1 و dict2 هر دو به یک مکان واحد در حافظه اصلی اشاره خواهند کرد یعنی هر دو دارای نشانی حافظه یکسان هستند به زبان ساده‌تر با اجرای دستور dict2=dict1 متغیر dict2 تنها یک ارجاع یا اشاره‌گر به dict1 است و یک dictionary مستقل نیست پس هر تغییری در dict1 به صورت خودکار در dict2 نیز بازتاب خواهد یافت. در پایتون هر تغییری شناسه یکتای خود را دارد که همان نشانی محل ذخیره سازی آن در حافظه است برای بدست آوردن شناسه یکتای هر متغیر از دستور id() استفاده می‌شود پس بنا برآنچه گفته شد در مثال زیر خروجی دستور id() برای هر دو واژه نامه dict1 و dict2 یکسان خواهد بود.

```
dict1 = {'brand':'Toyota', 'model':'prado', 'year':2009}
dict2 = dict1
print(id(dict1))
print(id(dict2))
```

برای چیره شدن بر این مشکل در کلاس dictionary شگردی به نام copy() پیش بینی شده است که وظیفه آن ایجاد یک واژه نامه مستقل از یک واژه نامه موجود است به گونه‌ای که عناصر واژه نامه جدید همان عناصر واژه نامه موجود است.

مثال :

```
CarDict = {'brand':'Toyota',  
           'model':'prado',  
           'year':2009}  
CarDict2 = CarDict.copy()  
print(CarDict2)  
# output {'brand': 'Toyota', 'model': 'prado', 'year': 2009}
```

روش دیگر رونوشت برداری از یک واژه نامه استفاده از سازنده کلاس Dictionary یعنی dict() است :

مثال :

```
CarDict = {'brand':'Toyota',  
           'model':'prado',  
           'year':2009}  
CarDict2 = dict(CarDict)  
print(CarDict2)  
# output {'brand': 'Toyota', 'model': 'prado', 'year': 2009}
```

واژه نامه های تو در تو

عناصر یک واژه نامه می توانند خود یک واژه نامه باشند. به این دسته از واژه نامه ها واژه نامه های تو در تو گفته می شود برای مثال در کد زیر واژه نامه Car_info در بردارنده اطلاعات چهار نوع خودرو است :

```
Car_info = {  
    'car1':{'brand':'Toyota',  
            'model':'prado',  
            'year':2009},
```

```
'car2':{'brand':'Hyundai',
'model':'venue',
'year':2023},
'car3':{'brand':'Peugeot',
'model':'207',
'year':2021},
}
```

به منظور افزایش خوانایی کد نوشته شده می توانید هر واژه نامه را جداگانه تعریف و مقداردهی کنید و سپس به عنوان مقدار در واژه نامه Car_info استفاده کنید :

```
brand1= {'brand':'Toyota',
'model':'prado',
'year':2009}
brand2 = {'brand':'Hyundai',
'model':'venue',
'year':2023}
brand3 = {'brand':'Peugeot',
'model':'207',
'year':2021}
car_info = {
'car1': brand1 ,
'car2': brand2,
'car3': brand3
}
print(car_info)
```

برای دسترسی به یک عنصر خاص از واژه نامه های تودرتو باید به تعداد عمق واژه نامه از عملگر انتخاب [] استفاده کنید. برای مثال دستور زیر مقدار Hyundai را در خروجی چاپ خواهد کرد.

```
print(car_info['car2']['brand'])
```

مثال :

```
student1 = {'name':'arash', 'family':'izanlou'}
student2 = {'name':'arman', 'family':'iranmanesh'}
student3 = {'name':'dariush', 'family':'keyvan'}

team1 = {'sport' : 'footbal', 'student':student1}
team2 = {'sport' : 'tennis', 'student':student2}
team3 = {'sport' : 'volleyball', 'student':student3}

school_teams = {
    'team1': team1 ,
    'team2': team2,
    'team3': team3
}

print(school_teams['team2']['student']['name'])
# output is : arman
```

فصل دوازدهم

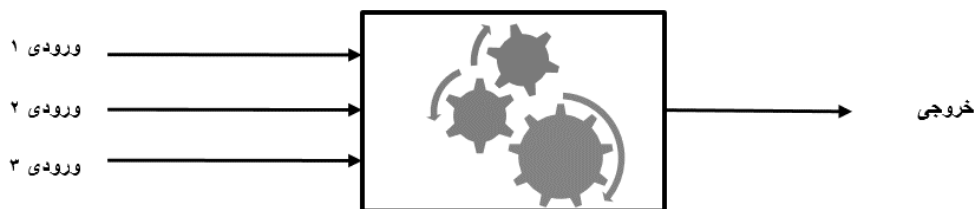
تابع

گاهی نیاز است تا قطعه کدی که یک وظیفه مشخص را انجام می دهد در بخشهای مختلف یک برنامه ، بارها اجرا شود. برای مثال در سامانه مدیریت تحصیلی دانشگاه ، قطعه برنامه محاسبه میانگین نمرات دانشجویان به هنگام پردازش کارنامه پایان ترم ، محاسبه میزان تخفیف شهریه و یا تعیین تعداد واحدهایی که دانشجویان می توانند در آخرین نیم سال تحصیلی بردارند اجرا می شود. نیاز به اجرای چند باره یک قطعه کد واحد در بسیاری از برنامه های بزرگ که برای حل مسائل دنیای واقعی نوشته می شوند وجود دارد. و باید به خوبی تدبیر گردد. از دیگر سو برای نوشتن یک برنامه بزرگ و پیچیده با چندین وظیفه مختلف می توان آن را به چند زیر برنامه کوچک که هر یک وظیفه مشخصی را انجام می دهند شکست و هر وظیفه را جداگانه کد نویسی و پیاده سازی کرد. این روش ، اشکال زدایی نرم افزار و آزمون درستی کارکرد آن را ساده تر می کند. برای مثال سامانه بانکداری ، یک نرم افزار بزرگ و پیچیده است که می تواند به بخش های کوچکتری چون : افتتاح حساب ، واریز ، برداشت ، انتقال ، حواله ، بستن حساب و دیگر وظایف موجود در حوزه بانکداری شکسته شود.

بنا بر آنچه گفته شد در زبان های برنامه نویسی باید راهی برای دوری گزیدن از نوشتن کدهای تکراری و نیز امکان تجزیه یک مسئله بزرگ و پیچیده به چندین مسئله کوچک و ساده تر وجود داشته باشد ، بایستگی برآوردن این دو نیاز، سرآغاز برآمدن یک مفهوم بسیار مهم و اساسی در زبانهای برنامه نویسی است که از آن با نام تابع (function) یاد می شود.

یک تابع ، قطعه برنامه کوچکی است که برای انجام یک کار مشخص نوشته می شود و دارای یک نام است و می تواند در بخش های مختلف برنامه اصلی با استفاده از نام آن ، بارها فراخوانی گردد. به بیان ساده تر ، تابع برنامه کوچکی است که قابلیت استفاده مجدد دارد یعنی برنامه نویس تنها یکبار آن را می نویسد. و بارها و بارها در بخش های مختلف برنامه اصلی و تنها با نوشتن یک نام

(به جای نوشتن دوباره قطعه کد) آن را فرا می خواند. یک تابع ، داده های مورد نیاز خود را از برنامه فراخوان دریافت می کند و با انجام پردازش روی آنها نتیجه به دست آمده را به برنامه فراخوان برمی گرداند. از این دیدگاه تابع همانند دستگاهی است که می تواند با انجام عملیات بر روی ورودی های خود ، یک خروجی تولید کند.



شکل ۱-۱۲ مفهوم تابع به مانند یک سامانه ورودی پردازش خروجی

پیامدهای سودمند بهره گیری از تابع در برنامه نویسی :

- ۱- پیشگیری از تکرار کد
- ۲- بهره گیری دوباره از کدی که تنها یک بار نوشته شده است.
- ۳- اشکال زدایی از کد و آزمون درستی کارکرد آن بهتر و سریع تر انجام می شود.
- ۴- نگهداری ساده تر از برنامه
- ۵- شکستن یک برنامه بزرگ و پیچیده به چندین برنامه کوچک و ساده تر
- ۶- مفهوم متغیرهای محلی در تابع ، مدیریت فضای نام برنامه را بسیار آسان می سازد.
- ۷- بهبود خوانایی برنامه و درک ساده تر آن
- ۸- ساخت یافتگی کد برنامه افزایش یافته و نظم درونی و ارتباط منطقی حاکم بر بخش های مختلف نرم افزار پیشینه می گردد.

نوشتن تابع در پایتون

در زبان برنامه نویسی پایتون، یک تابع با رهنمون `def` تعریف می شود و دارای چهار بخش اصلی است: نام تابع، ورودی های تابع، بدنه تابع و خروجی تابع

۱- نام تابع

نام تابع که پس از رهنمون `def` قرار می گیرد. باید از قوانین نامگذاری متغیرها پیروی کند و بهتر است بیانگر هدف تابع و نشان دهنده کاری باشد که تابع انجام می دهد. برای مثال اگر کار تابع رسم یک دایره است نام `DrawCircle` زیبنده آن خواهد بود. همیشه پس از نام تابع یک جفت کمانک باز و بسته () قرار خواهد گرفت. در زبان برنامه نویسی پایتون فراخوانی یک تابع با استفاده از نام تابع به همراه کمانک بازوبسته پس از آن انجام می شود.

مثال: بخش نام در تابعی که مساحت یک مستطیل را محاسبه می کند می تواند همانند الگوی زیر باشد:

```
def calc_rect_erea ()
```

۲- ورودی های تابع

به داده های خام مورد نیاز تابع که باید توسط برنامه فراخوان در دسترس تابع قرارگیرند ورودی های تابع گفته می شود. یک تابع می تواند هیچ و یا چندین ورودی داشته باشد. ورودی های مورد نیاز یک تابع پس از نام تابع و درون جفت کمانک باز و بسته () قرار می گیرند. در فرهنگ واژگان زبان های برنامه نویسی به ورودی های یک تابع که به هنگام تعریف آن و در قالب یک یا چند متغیر، مشخص می شوند ورودی های ظاهری یا پارامتر^۱ گفته می شود و به داده هایی که به هنگام فراخوانی تابع به آن فرستاده

¹ Parameter

می شوند و جانشین پارامترهای تعریف شده در تابع می شوند آرگومان^۱ یا ورودی واقعی گفته می شود.

با وجود آنکه بسیاری از برنامه نویسان واژه های Parameter و Argument را به جای یکدیگر بکار می برند اما چنان که گفته شد هریک از این دو متفاوت با دیگری است و بکارگیری این دو در معنای یکسان شایسته یک برنامه نویس خبره نیست. در حقیقت ورودی های مجازی یا پارامتر به متغیرهایی گفته می شود که در زمان تعریف یک تابع درون جفت کمانک بازوبسته پس از نام آن قرار می گیرند در حالی که ورودی واقعی یا آرگومان اشاره به داده هایی دارد که به هنگام فراخوانی تابع، در دسترس تابع قرار گرفته و جانشین پارامترهای تعریف شده در آن می شوند.

مثال : تابع `calc_rect_era` () که وظیفه محاسبه مساحت یک مستطیل را دارد نیاز به دانستن طول و عرض مستطیل دارد بنابراین برنامه فراخوان باید طول و عرض مستطیل مورد نظر را در دسترس این تابع قرار دهد پس این تابع باید دو ورودی داشته باشد که یکی طول مستطیل (`length`) است و دیگری عرض مستطیل (`width`) از این رو تا این جای کار تعریف تابع همانند زیر خواهد بود :

```
def calc_rect_era (length , width)
```

مثال : تابع `get_now()` که تاریخ و زمان جاری را بر می گرداند نیاز به هیچ ورودی ندارد از این رو الگوی بخش نام و پارامترهای آن همانند زیر خواهد بود :

```
def Get_now()
```

مثال : تابع `MiladyToShamsi()` که وظیفه تبدیل تاریخ میلادی به شمسی را بر عهده دارد به سه ورودی سال میلادی ، ماه میلادی و روز نیاز دارد پس الگوی نام و پارامترهای آن همانند زیر خواهد بود :

```
MiladyToShamsi(year,month,day)
```

¹ Argument

۳- بدنه تابع

برنامه کوچکی که وظیفه اصلی تابع را انجام می دهد در بدنه تابع قرار می گیرد. کدهای موجود در بدنه تا زمانی که تابع فراخوانی نگردد اجرا نخواهند شد. دستورات موجود در بدنه تابع پس از علامت دو نقطه بیانی : قرار می گیرند و باید با استفاده از یک و یا چندین فاصله نسبت به رهنمون `def` تورفتگی داشته باشند مقدار تورفتگی را برنامه نویس به دلخواه انتخاب می کند اما باید تا پایان قطعه کد بدنه ثابت باقی بماند بیشتر برنامه نویسان برای ایجاد تورفتگی از چهار فاصله خالی استفاده می کنند. بدنه تابع با اولین تورفتگی نسبت به رهنمون `def` آغاز و با اولین دستور بدون تورفتگی پایان می یابد

مثال : می دانیم که مساحت یک مستطیل برابر است با حاصلضرب اندازه طول آن در اندازه عرض آن پس کدی که در بدنه تابع `() calc_rect_erea` قرار می گیرد همانند الگوی زیر خواهد بود :

```
def calc_rect_erea (length , width) :
```

```
    erea = length * width
```

در پایتون بدنه یک تابع هرگز نمی تواند خالی باشد از این رو چنانچه به هر دلیلی نیاز به داشتن چنین تابعی دارید برای پیشگیری از اعلام خطا توسط مفسر پایتون می توانید از یک دستور `pass` در بدنه تابع استفاده کنید. این دستور هیچ کاری انجام نمی دهد اما از دید مفسر پایتون یک دستور به شمار می رود و در نتیجه مانع از اعلام خطا می گردد.

مثال :

```
def myfunction():
```

```
    pass
```

۴- خروجی تابع

یک تابع می تواند هیچ خروجی نداشته باشد و یا تنها یک خروجی داشته باشد. خروجی تابع با دستور `return` که در بدنه تابع قرار می گیرد به برنامه فراخوان برگردانده می شود. مفسر پایتون با رسیدن به دستور `return` خروجی تابع را به برنامه فراخوان تحویل داده و به اجرای تابع پایان می دهد. با پایان یافتن اجرای تابع، اجرای برنامه اصلی از نخستین خط پس از تابع از سر گرفته می شود.

دستور `return` به تنهایی یعنی زمانی که مقدار یا عبارتی پس از آن نوشته نشود مقدار `None` را برمی گرداند همچنین اگر در انتهای تابع دستور `return` نوشته نشود، مفسر پایتون به صورت ضمنی دستور `return None` را برای آن در نظر خواهد گرفت. بنابراین با فراخوانی این گونه توابع و پس از اجرای کامل دستورات داخل بدنه مقدار `None` بازگردانده خواهد شد.

مثال : در تابع `calc_rect_area()` خروجی تابع مساحت محاسبه شده یک مستطیل است پس بدنه تابع که دارای دستور `return` است همانند زیر نوشته می شود :

def calc_rect_area (length , width) :

area = length * width

return area

مثال : تابع زیر تنها یک پیام "Hello World" چاپ خواهد کرد بنابراین هیچ خروجی به برنامه فراخوان برنمی گرداند از این رو نیازی به دستور `return` ندارد :

def say_hello ():

print("Hello World ")

فراخوانی یک تابع

در زبان برنامه نویسی پایتون فراخوانی یک تابع با استفاده از نام تابع و یک جفت کمانک بازویسته () که پس از نام تابع قرار می گیرد انجام می شود.

مثال : در برنامه زیر ابتدا طول و عرض یک مستطیل از کاربر گرفته می شود و سپس ورودی های فراهم شده توسط کاربر در دسترس تابع `calc_rect_erea()` قرار می گیرند تا مساحت مستطیل محاسبه گردد و سرانجام خروجی این تابع در متغیر `RectArea` قرار گرفته و با چاپ مقدار این متغیر برنامه اصلی پایان می یابد همانگونه که در کد برنامه دیده می شود تابع `calc_rect_erea()` دارای دو پارامتر به نام های `length` و `width` است که به هنگام تعریف تابع مشخص شده اند و در زمان فراخوانی دو آرگومان `w` و `h` به تابع ارسال شده است :

```
def calc_rect_erea (length , width) :
```

```
    area = length * width
```

```
    return area
```

```
w = float(input("pleas Enter Rect width :"))
```

```
h = float(input("pleas Enter Rect length:"))
```

```
RectArea = calc_rect_erea (w , h)
```

```
print(RectArea)
```

نکاتی چند در باره پارامتر و آرگومان :

۱- تعداد آرگومان های ارسالی به یک تابع در هنگام فراخوانی آن باید برابر با تعداد پارامترهایی باشد که در زمان تعریف تابع مشخص شده اند وگرنه مفسر پایتون اعلام خطا خواهد کرد.

مثال : در کد زیر تابع `add()` دارای سه پارامتر `a` , `b` , `c` است اما به هنگام فراخوانی تنها دو آرگومان دریافت کرده است بنابراین مفسر پایتون خطای زیر را اعلام می کند :

```
missing 1 required positional argument: c
```

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

```
s = a + b + c
```

```
return s
```

```
num1 = 10
```

```
num2 = 15
```

```
num3 = 5
```

```
rslt = add (num1, num2)
```

```
print(rslt)
```

۲- ارسال آرگومانها به تابع باید به همان ترتیب قرار گرفتن پارامتر ها در تعریف تابع باشد برای مثال اگر تابع `add()` همانند زیر تعریف شود مفسر پایتون به هنگام فراخوانی تابع `add()` نخستین آرگومان را جایگزین پارامتر `a` ، دومین آرگومان را جایگزین پارامتر `b` و سومین آرگومان را جایگزین پارامتر `c` کرده و سپس کد موجود در بدنه تابع را اجرا می کند.

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

```
return a + b + c
```

```
s = add (۱۲ , ۷ , ۵)
```

```
print ( s )
```

آرگومانهای نامحدود

گاهی تعداد آرگومان هایی که می توان به هنگام فراخوانی یک تابع به آن ارسال کرد نمی تواند ثابت باشد و براساس شرایط برنامه و نیاز کاربرممکن است تغییر کند برای مثال آرگومان های تابع `weight_average()` که میانگین وزن دانش آموزان پایه دوم ابتدایی مدارس موجود در مناطق روستایی دارای کمتر از ۳۵ خانوار را محاسبه می کند نمی تواند ثابت باشد چرا که تعداد دانش آموزان پایه دوم ابتدایی از یک مدرسه تا مدرسه دیگر متفاوت است.

درپایتون برای فرستادن تعداد نامشخصی آرگومان به یک تابع ، باید به هنگام تعریف تابع یک علامت ستاره (*) پیش از نام پارامتری که می خواهید آرگومان های نامحدود ارسال شده به تابع جایگزین آن شوند قرار دهید. با این کار به مفسر پایتون می گوییم که برنامه فراخوان می تواند تعداد نامحدودی آرگومان به این تابع ارسال کند. درحقیقت دراین حالت مفسر پایتون آرگومان های ارسال شده به تابع را در قالب ساختمان داده tuple جایگزین پارامتر مورد نظر می کند بنابراین در بدنه تابع می توان با پارامتر یاد شده همانند یک tuple رفتار کرد برای مثال جهت دسترسی به هر یک از آرگومان های نامحدود کافی است شاخص آرگومان مورد نظر درون عملگر انتخاب (جفت علامت []) پس از نام پارامتر قرارگیرد و برای یافتن تعداد آرگومان های ارسال شده به تابع کافی است از دستور len(paramname) استفاده کنید.

مثال : تابعی بنویسید که بتواند تعداد نامحدودی آرگومان دریافت کرده و تعداد آرگومان های دریافتی را به برنامه فراخوان برگرداند.

```
def myfunc (*myparam) :
```

```
    count = len(myparam)
```

```
    return count
```

```
print (myfunc (5,3,10,14,15))
```

```
print (myfunc (7, 8))
```

```
print (myfunc (71, 18, 15, 14, 19,33 ,62,10, 19, 15.8, 16.7))
```

مثال : تابعی بنویسید که بتواند تعداد نامحدودی آرگومان عددی دریافت کرده و حاصل جمع آرگومان اول و دوم خود را به برنامه فراخوان برگرداند.

```
def myfunc (*mynumbers) :
```

```
    return mynumbers [0] + mynumbers [1]
```

```
print (myfunc (5,3,10,14,15))
```

```
print (myfunc (7, 8))
```

```
print (myfunc (71, 18, 15, 14, 19,33 ,62,10, 19, 15.8, 16.7))
```

مثال : تابعی بنویسید که بتواند میانگین تعداد نام مشخصی عدد که به عنوان ورودی به آن داده می شوند را محاسبه کند.

```
def numbers_average (*number):
```

```
    count = len(number)
```

```
    s = .
```

```
    for i in range(count):
```

```
        s += number[i]
```

```
    avrg = s / count
```

```
    return avrg
```

```
print (numbers_average (۵,۳,۱۰,۱۴,۱۵))
```

```
print (numbers_average (۷, ۸))
```

```
print (numbers_average (۷۱, ۱۸, ۱۵, ۱۴, ۱۹,۳۳ ,۶۲,۱۰, ۱۹, ۱۵.۸, ۱۶,۷))
```

ارسال آرگومانها با استفاده از نام پارامتر

در حالت عادی ، ارسال آرگومانها به تابع باید به همان ترتیب قرار گرفتن پارامتر ها در تعریف تابع باشد برای مثال اگر تابع `add()` همانند زیر تعریف شود مفسر پایتون به هنگام فراخوانی تابع `add()` نخستین آرگومان را جایگزین پارامتر `a` ، دومین آرگومان را جایگزین پارامتر `b` و سومین آرگومان را جایگزین پارامتر `c` کرده و سپس کد موجود در بدنه تابع را اجرا می کند.

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

```
    return a + b + c
```

```
s = add (۱۲ , ۷ , ۵)
```

```
print ( s )
```

در پایتون می توان آرگومان های یک تابع را با استفاده از نام پارامتر مورد نظر به تابع ارسال کرد در این روش نیازی به رعایت ترتیب قرار گرفتن پارامترها نیست. به این

دسته از ورودی های واقعی^۱ آرگومانهای نامدار گفته می شود. شیوه کلی نگارش و بکار گیری آرگومانهای نامدار به هنگام فراخوانی یک تابع همانند الگوی زیر است :

FunctionName(ParameterName = Argument)

مثال :

```
def add (a, b, c) :
    return a + b + c
s = add (b = 14 , a = 7 , c = 15)
print ( s )
```

مثال : تابع محاسبه حجم یک مکعب و ارسال آرگومان به روش با نام به آن :

حجم مکعب حاصلضرب طول (length) در عرض (width) در ارتفاع (height) آن است

```
def cube_mass (length, width, height) :
    return length * width * height
cube = cube_mass (height = 8 , length = 10 , width = 7)
print ( cube )
```

به هنگام فراخوانی توابع می توان همزمان از دو روش ارسال آرگومان با نام و عادی (ترتیبی) استفاده کرد به شرطی که آرگومان های ترتیبی پیش از آرگومان های نامدار و به همان ترتیب قرار گیری در تعریف تابع ارسال شوند.

مثال : فراخوانی تابع cube_mass() که در مثال قبلی تعریف شده است با ترکیبی از آرگومان های نامدار و عادی

```
cube = cube_mass ( 6 , 10 , height = 7)
```

¹ Argument

پارامترهای نامدار نامحدود

گاهی تعداد آرگومان های نامداری که می توان به هنگام فراخوانی یک تابع به آن ارسال کرد نمی تواند ثابت باشد و براساس شرایط برنامه و نیاز کاربر ممکن است تغییر کند برای مثال آرگومان های ارسالی به تابع `student_info_process()` که وظیفه پردازش اطلاعات شناسنامه ای دانش آموزان ابتدایی مدارس شهرستان بجنورد را دارد نمی تواند ثابت باشد چرا که ثبت اطلاعاتی چون نام ، نام خانوادگی ، نام پدر و شناسه ملی دانش آموز در پرونده تحصیلی اجباری است اما ثبت اطلاعاتی مانند شغل پدر ، وزن و قد دانش آموزان اجباری نیست. و از سویی هر مدرسه به فراخور برنامه ها و امکانات خود می تواند اطلاعات بیشتری در مورد هر دانش آموز در پرونده تحصیلی ثبت و نگهداری کند.

در پایتون برای ارسال تعداد نامشخصی آرگومان نامدار به یک تابع ، باید به هنگام تعریف تابع یک علامت دوستاره (*) پیش از نام پارامتر مورد نظر قرار دهید. با این کار به مفسر پایتون می گوئیم که برنامه فراخوان می تواند تعداد نامحدودی آرگومان نامدار به این تابع ارسال کند. در حقیقت در این حالت مفسر پایتون آرگومان های نامدار ارسال شده را در قالب ساختمان داده `dictionary` به تابع ارسال می کند بنابراین در بدنه تابع می توان با پارامتر یاد شده همانند یک `dictionary` رفتار کرد

مثال :

```
def student_info_process(studentid, **kwargs):
    for k in kwargs:
        print( k + " : " + kwargs[k])
student_info_process(100, Name="Arash", Family="Izanlou" ,
    Father="Mohammad" ,Code="0681234567")
```

خروجی کد بالا همانند زیر است :

```
Name : Arash
Family : Izanlou
Father : Mohammad
Code : 0681234567
```

تا اینجا می دانیم که هرگاه برنامه ما تابعی را برای اجرا فراخوانی کند مفسر پایتون دو روش برای انطباق آرگومان های ارسالی به تابع با پارامترهای تعریف شده در آن در اختیار دارد :

روش عادی یا ترتیبی که ارسال آرگومانها به تابع باید به همان تعداد و به ترتیب قرار گرفتن پارامتر ها در تعریف تابع باشد و نیز روش آرگومان های نامدار که آرگومان های یک تابع با استفاده از نام پارامتر مورد نظر و به صورت مقدار= نام پارامتر به تابع ارسال می شوند در این روش نیازی به رعایت ترتیب قرار گرفتن پارامترها نیست.

همچنین می دانیم که به هنگام فراخوانی توابع می توان ترکیبی از آرگومان های نامدار و عادی (ترتیبی) را مطابق تعریف تابع به آن ارسال کرد. به شرط آنکه آرگومان های عادی پیش از آرگومان های نامدار و به همان ترتیب قرار گیری در تعریف تابع ارسال شوند.

می توان برنامه فراخوان را وادار کرد تا ارسال آرگومان به برخی از پارامترهای یک تابع را فقط به شیوه آرگومان های نامدار انجام دهد یعنی ارسال آرگومان به آنها به روش عادی (ترتیبی) غیر فعال گردد به این منظور باید به هنگام تعریف تابع و پیش از نام پارامترهای مورد نظر از نویسه * به عنوان یک پارامتر نشانگر استفاده کرد در این حالت تمامی آرگومانهای متناظر با پارامترهای بعد از * باید به صورت نامدار یعنی به شکل مقدار = نام پارامتر ارسال شوند.

همچنین می توان برنامه فراخوان را وادار ساخت تا ارسال آرگومان به برخی از پارامترهای یک تابع را فقط به روش عادی (ترتیبی) یعنی بر اساس موقعیت پارامتر های آن انجام دهد یعنی امکان ارسال آرگومان به روش نامدار و به شکل مقدار= نام پارامتر برای آنها غیرفعال شود. برای دست یابی به این هدف باید به هنگام تعریف تابع و پس از نام پارامترهای مورد نظر نویسه / را به عنوان یک پارامتر نشانگر وارد کنیم. برای مثال در تابع f که به شکل زیر تعریف شده است آرگومان های ارسالی به پارامترهای pos1 و pos2 باید به صورت عادی (ترتیبی) باشند اما آرگومان ارسالی به پارامتر pos_or_kwd می تواند به هر دو صورت عادی و نامدار باشد درحالی که آرگومان های متناظر با هریک از پارامترهای kwd1 و kwd2 تنها باید به شکل نامدار ارسال شوند.

```
def f(pos1, pos2, /, pos_or_kwd, *, kwd1, kwd2):
```

```
    pass
```

مثال :

```
def standard_arg(arg):
    print(arg)
def pos_only_arg(arg, /):
    print(arg)
def kwd_only_arg(*, arg):
    print(arg)
def combined_example(pos_only, /, standard, *, kwd_only):
    print(pos_only, standard, kwd_only)

standard_arg(2)
standard_arg(arg=2)
combined_example(1, 2, kwd_only=3)
combined_example(1, standard=2, kwd_only=3)
```

پارامترهای دارای مقدار پیش فرض

در پایتون هر پارامتر می تواند یک مقدار پیش فرض داشته باشد ، این مقدار به هنگام تعریف تابع و با استفاده از عملگر انتساب (=) به پارامتر مورد نظر داده می شود. و چنانچه به هنگام فراخوانی تابع ، آرگومانی به این پارامتر ارسال نگردد مفسر پایتون همان مقدار پیش فرض را جایگزین پارامتر خواهد کرد.

مثال : دربرنامه زیر هر سه پارامتر تابع `cube_mass()` دارای مقدار پیش فرض هستند.

```
def cube_mass (length = 10 , width = 8, height = 5) :

    return length * width * height

cube = cube_mass (height = 8 , length = 10 , width = 7)
```

```
print ( cube )

cube = cube_mass (۸, ۱۱ , ۹)

print ( cube )

cube = cube_mass (۱۵)

print ( cube )

cube = cube_mass (width = 6)

print ( cube )

cube = cube_mass ()

print ( cube )
```

در باره پارامترهای پیش فرض دو نکته بسیار مهمی که لازم است همواره به یاد داشته باشید این است که :

۱- مقدار پیش فرض در نقطه تعریف تابع ارزیابی می شود و نه به هنگام فراخوانی آن

مثال : برای مثال کد زیر عدد ۵ را چاپ خواهد کرد نه عدد ۶ را

```
i = 5
def func(arg=i):
    print(arg)

i = 6
func()
```

۲- مقدار پیش فرض تنها یک بار ارزیابی می شود.

مثال :

```
def f(a, L=[]):  
    L.append(a)  
    return L  
print(f(1))  
print(f(2))  
print(f(3))
```

خروجی مثال بالا همانند زیر است :

[1]

[1, 2]

[1, 2, 3]

فصل سیزدهم

فضای نام و دامنه

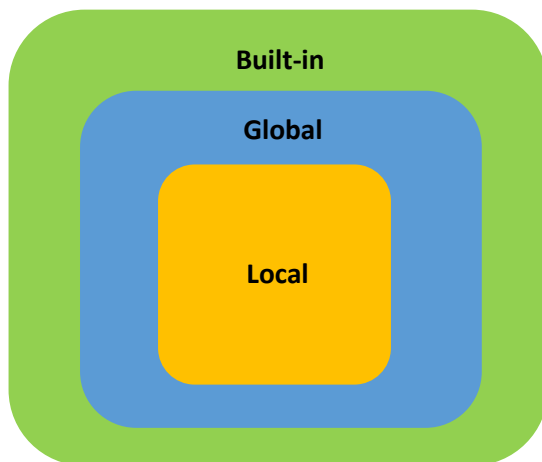
هر برنامه پایتون در بردارنده تعداد زیادی از عناصر برنامه نویسی مانند متغیر، تابع، کلاس و شی است که باید به هر یک نامی داده شود تا مفسر پایتون با استفاده از آن بتواند عنصر را شناسایی و به آن دست یابد. به این نام شناسه^۱ عنصر گفته می شود. برای مثال در دستور جایگزینی $\text{Age} = 25$ می توان گفت که ۲۵ یک شی ذخیره شده در حافظه است و Age نامی است که به آن نگاشت شده است. به زبان ساده تر در یک برنامه پایتون هر عنصر برنامه نویسی دارای یک نام است که از آن برای شناسایی عنصر و دستیابی به آن استفاده می شود. از این رو وجود شیوه های کارآمد و موثر جهت تضمین یکتایی نام ها و چگونگی برخورد با نام های مشابه، ضرورتی گریزناپذیر است. به این منظور پایتون از مفهومی به نام فضای نام^۲ برای ساماندهی، دسته بندی و اداره نام های موجود در یک برنامه و تضمین یکتایی آنها و چگونگی رفتار با نام های مشابه استفاده می کند.

فضای نام مجموعه ای از شناسه ها (نامها) است که در آن هر نام به یک شی^۳ نگاشت شده است به زبان ساده تر فضای نام نگاشتی از نامها به اشیاء است. فضای نام در پایتون یک مفهوم منطقی است که به صورت تودرتو و در سه سطح سازماندهی می شود هر سطح در زمان خاصی ساخته می شود و طول عمر معینی دارد. شکل زیر مفهوم تودرتو بودن فضای نام در پایتون را به تصویر می کشد.

^۱ Identifier

^۲ Name Space

^۳ Object



تصویر ۱۳-۱ مفهوم فضای نام در پایتون

در بالاترین سطح ، فضای نامی ذاتی (Built-in) قرار دارد این سطح با شروع مفسر پایتون به صورت خودکار ساخته می شود و تا زمانی که مفسر در حال اجراست وجود خواهد داشت. به همین دلیل است که توابعی چون `print()` ، `input()` و دیگر توابع و دستورات پیش ساخته^۱ پایتون همیشه و در هر بخشی از برنامه در دسترس هستند.

دومین سطح که درون سطح اول قرار دارد فضای نام سراسری (Global) نامیده می شود. این سطح وابسته به یک فایل در بردارنده کد پایتون است یعنی با ایجاد هر فایل دارای پسوند `.py` فضای نام سراسری ویژه آن فایل هم ساخته می شود. لازم به یاد آوری است که برنامه های نوشته شده با پایتون در فایل هایی با پسوند `.py` ذخیره می شوند. هر فایل فضای نام سراسری ویژه خود را دارد که جدای از فضای نام سراسری دیگر فایل هاست ، بنابراین وجود نام های یکسان در فایل های جداگانه مجاز است.

در فرهنگ واژگان پایتون به فایل های در بردارنده دستورات ، ساختارهای برنامه نویسی و دیگر عناصر زبان که مستقیم اجرا می شوند اسکریپت (Script) و به فایل های در بردارنده عناصر ساختارهایی چون متغیرها ، توابع و کلاس ها که به طور مستقیم اجرا نمی شوند و می توان کدهای

¹ Built-in

نوشته شده در آنها را با استفاده از رهنمون `import` در یک فایل دیگر وارد و سپس فراخوانی و اجرا کرد پیمانه (Module) گفته می شود.

فضای نام سراسری هر فایل به طور کامل جدای از هر فایل دیگری است یعنی نمی توان از یک فایل به نام های موجود در دیگری دست یافت بنابراین برای دسترسی به نام های درون فضای نام سراسری مورد نیاز باید آن را به درون فایل مورد نظر خود وارد کنید که این کار با استفاده از رهنمون `import` انجام می شود. برای مثال فرض کنید برنامه ما دارای دو فایل به نام های `calculate.py` و `main.py` است در فایل `calculate.py` تابعی به نام `Fibo()` و به شکل زیر تعریف کرده ایم که می تواند `n` جمله اول سری فیبوناتچی را تولید و چاپ کند.

```
def Fibo(n):
    a, b = 0, 1
    while(n>0):
        print(a, sep=" ")
        a, b = b, a + b
        n -= 1
```

به صورت معمول این تابع از درون فایل `main.py` قابل دسترس نیست و استفاده از آن در این فایل بدون استفاده از رهنمون `import` موجب اعلام خطای زیرتوسط مفسر پایتون خواهد شد :

NameError: name 'Fib' is not defined

علت اعلام خطا این است که هر دو فایل `calculate.py` و `main.py` دارای فضای نام سراسری ویژه خود هستند که به طور کامل جدای از یکدیگرند و به هم دسترسی ندارند. برای حل این مشکل کافی است همانند زیر رهنمون `import` را بکار گیرید در این کد چگونگی دستیابی به شناسه `Fibo` که به یک تابع از فضای نام `calculate` اشاره دارد نشان داده شده است :

```
import calculate
calculate.Fibo(7)
```


و سرانجام درونی ترین سطح ، فضای نام محلی (local) است که درون سطح سراسری قرار می گیرد. یک فضای نام محلی در زمان فراخوانی یک تابع ساخته می شود. و در بردارنده تمامی نام های موجود در بدنه تابع و نام پارامترهای آن است. به زبان ساده تر با فراخوانی هر تابع یک فضای نام محلی برای آن تابع درون فضای نام سراسری ساخته می شود و با پایان یافتن اجرای تابع از بین می رود. به همین دلیل است که می توان درون یک تابع متغیرهایی همنام با متغیر های خارج از تابع تعریف کرد. زیرا آنها در دو فضای نام جداگانه قرار دارند. و چون فضای نام محلی درون فضای نام سراسری ساخته می شود می توان از درون یک تابع به متغیرهای خارج از آن که در فضای نام سراسری هستند دسترسی پیدا کرد.

دامنه یا قلمرو یک نام

وجود فضاهای نام چندگانه و گوناگون به این معنی است که همزمان چندین نمونه مختلف از یک نام واحد می توانند در یک برنامه پایتون وجود داشته باشند و تا هنگامی که هر نمونه در یک فضای نام جداگانه زندگی می کند با یکدیگر هیچ گونه برخورد و تداخلی نخواهند داشت. برای مثال نام Average می تواند در چندین پیمانه (Module) جداگانه وجود داشته باشد. اما اگر در یک پیمانه واحد نام های مشابه وجود داشته باشند پایتون چگونه آنها را اداره می کند به بیان ساده تر پرسش این است که ساماندهی و اداره کردن نام ها درون یک فایل واحد چگونه است. به ویژه آنکه می دانیم درون یک فایل جایی است که فضاهای نام محلی دیگری همچون توابع یا کلاس ها نیز می توانند وجود داشته باشند و برای تمیز آنها از یکدیگر پیشوند خاصی همانند آنچه رهنمون import در اختیارمان می گذاشت وجود ندارد. پاسخ این پرسش در مفهومی به نام محدوده یا دامنه نهفته است. دامنه یا قلمرو یک نام ، ناحیه یا بخشی از متن کد برنامه است که در صورت نیاز و بدون استفاده از هیچ پیشوندی می توان به آن نام دسترسی داشت و از آن استفاده کرد. بیرون از این ناحیه چنین نامی وجود ندارد و نمی توان به آن دست یافت.

دامنه محلی

بدنه و سرآیند هر تابع یک حوزه محلی^۱ است. سرآیند یک تابع همان فضای بین دو کمانک بازوبسته پس از نام تابع است که در آن ورودی های ظاهری^۲ تابع قرار می گیرند از این رو در زبان برنامه نویسی پایتون هر متغیری که درون یک تابع تعریف شود از دید مفسر پایتون یک متغیر محلی است و دامنه دسترسی به آن محدود به همان تابع است. و بیرون از آن تابع نمی توان به آن دسترسی داشت. بر اساس آنچه که گفته شد دامنه ورودی های ظاهری تعریف شده در تابع هم محلی است.

برای مثال در کد زیر مفسر پایتون با رسیدن اجرای کد به دستور `print(x)` اعلام خطا خواهد کرد چرا که متغیر `x` درون تابع `square()` تعریف شده و دامنه دسترسی به آن محلی است پس با خروج از تابع از بین می رود بنابراین بیرون از تابع ناشناخته است و نمی توان به آن دسترسی داشت.

```
def square:()
```

```
    x=5
```

```
    print(x * x)
```

```
square()
```

```
print(x)
```

```
#output NameError: name 'x' is not defined
```

در توابع تودرتو یعنی حالتی که درون یک تابع ، تابع دیگری تعریف می شود تابع درونی به متغیر تعریف شده در تابع بیرونی دسترسی دارد اما عکس آن ممکن نیست یعنی از تابع بیرونی نمی توان به متغیر محلی تعریف شده در تابع درونی دست یافت. برای نمونه در مثال زیر متغیر `n` درون تابع `calc()` تعریف شده است بنابراین تابع بیرونی `square()` به این متغیر دسترسی ندارد اما تابع `calc()` که درون تابع `square()` تعریف شده است می تواند به متغیر `x` که در تابع بیرونی `square()` تعریف شده دست یابد.

¹ Local

² Parameters

مثال :

def square() :

x = 5

def calc():

n = x

return n * n

print(calc())

square()

متغیر غیر محلی

در تابع های تودرتو در حالت عادی هرمتغیری که درون یک تابع داخلی^۱ تعریف می شود از دید مفسر پایتون یک متغیر محلی است و دامنه دسترسی به آن محدود به همان تابع است. با این همه گاهی نیاز است که درون یک تابع داخلی، مقدار متغیر تعریف شده در تابع بیرونی^۲ تغییر داده شود. این کار در حالت عادی ممکن نیست چرا که با دادن مقدار جدید به متغیر مورد نظر درون بدنه تابع داخلی، یک متغیر جدید محلی همانام با متغیر موجود در تابع بیرونی ساخته می شود و مقدار جدید به آن داده می شود از این رو مقدار متغیر موجود در تابع بیرونی بدون تغییر باقی می ماند. برای چیرگی بر این چالش می توانید از رهنمون nonlocal پیش از نام متغیر دلخواه خود استفاده کنید. در یک تابع درونی، بکار گیری رهنمون nonlocal پیش از نام یک متغیر بدین معنی است که دامنه آن نه محلی است و نه سراسری بلکه محل زندگی آن در نزدیکترین محدوده در برگیرنده دامنه محلی جاری است. که در حالت توابع تودرتو نزدیکترین محدوده در برگیرنده دامنه محلی (تابع درونی) همان دامنه محلی تابع بیرونی است.

مثال: ایجاد یک متغیر در تابع درونی که همانام با متغیر موجود در تابع بیرونی است :

¹ inner

² outer

```
def outer_func() :
    x=5
    def inner_func():
        x=7
        return x * x
    print(inner_func())
    print(x)
outer_func()
```

خروجی دستور بالا به ترتیب ۴۹ و ۵ است

بازنویسی همان مثال با استفاده از رهنمون `nonlocal` :

```
def outer_func():
    x=5
    def inner_func():
        nonlocal x
        x = 7
        return x * x
    print(inner_func())
    print(x)
outer_func()
```

خروجی دستور بالا ۴۹ و ۷ خواهد بود. همان گونه که دیده می شود تابع درونی مقدار متغیر تعریف

شده در تابع بیرونی را تغییر داده است.

دامنه سراسری

به نواحی درون یک فایل که خارج از بدنه هر تابع یا کلاسی که درون آن فایل تعریف شده است قرار دارند دامنه یا حوزه سراسری^۱ گفته می شود. بسیار مهم است بدانید که واژه سراسری در بحث فضای نام و حوزه تنها به سرتاسر یک فایل اشاره می کند و نه به سراسری یک برنامه.

^۱ Global Scope

متغیر سراسری

به متغیری که درون یک فایل و خارج از هر تابع یا کلاسی تعریف می شود یک متغیر عمومی یا سراسری^۱ گفته می شود. برای نمونه در مثال زیر متغیر `msg` یک متغیر سراسری است و از آنجا که هر تابعی می تواند به متغیر سراسری دسترسی داشته باشد از این رو دو تابع `sayhello()` و `printmessage()` بکار گرفته شده است :

```
msg="Hellow Wrld"
def sayhello():
    print(msg)
def printmessage(mymsg):
    print(mymsg)
sayhello()
printmessage(msg)
```

چنانچه در تابعی یک متغیر محلی همنام با یک متغیر سراسری وجود داشته باشد مفسر پایتون این متغیر را محلی در نظر خواهد گرفت و هر تغییری در مقدار آن تاثیری بر مقدار متغیر سراسری همنام آن نخواهد داشت. متغیر سراسری همنام همچنان سراسری است و مقدار خود را حفظ خواهد کرد. برای مثال در کد زیر در تابع `sayhello ()` متغیری با نام `msg` همنام با متغیر سراسری `msg`

تعریف شده است

```
msg = "Hellow Global World"
def sayhello ():
    msg = "Hellow Local World"
    print(msg)
```

^۱ global variable

```
def printmessage(mymsg):
```

```
    print(mymsg)
```

```
sayhello() #output is : Hellow Local World
```

```
printmessage(msg) # output is : "Hellow Global World"
```

در حالت عادی هر متغیری که درون یک تابع تعریف شود از دید مفسر پایتون یک متغیر محلی است و دامنه دسترسی به آن محدود به همان تابع است. با این همه اگر به هر دلیلی نیاز دارید که درون یک تابع یک متغیر عمومی تعریف کنید می توانید از رهنمون `global` پیش از نام متغیر دلخواه خود استفاده کنید. در یک تابع، بکار گیری رهنمون `global` پیش از نام یک متغیر دامنه دسترسی به آن متغیر را از محلی به سراسری تغییر خواهد داد.

مثال :

```
def sayhello():
```

```
    global msg
```

```
    msg="Hellow World"
```

```
    print(msg)
```

```
def printmessage(mymsg):
```

```
    print(mymsg)
```

```
sayhello() # output is : Hellow World
```

```
printmessage(msg) # output is : Hellow World
```

```
print(msg) # output is : Hellow World
```

کاربرد دیگر رهنمون `global` تغییر مقدار یک متغیر سراسری از درون یک تابع است. برای این کار در تابع مورد نظر، باید متغیر سراسری را با استفاده از رهنمون `global` دوباره تعریف کنید و گرنه همانگونه که پیشتر گفته شد چنانچه در تابعی یک متغیر محلی همانم با یک متغیر سراسری وجود داشته باشد مفسر پایتون این متغیر را محلی در نظر خواهد گرفت و هر تغییری در مقدار آن تاثیری بر مقدار متغیر سراسری همانم آن نخواهد داشت.

مثال :

```
msg = "Hello World"
```

```
def sayhello():
```

```
    global msg
```

```
    msg="Hello My Country"
```

```
    print(msg)
```

```
def printmessage(mymsg):
```

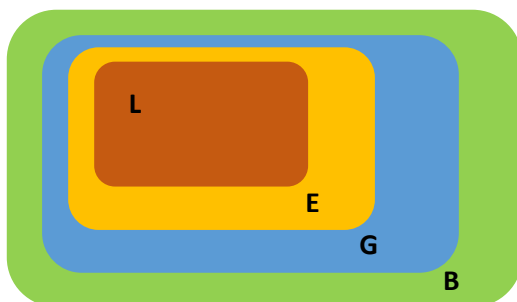
```
    print(mymsg)
```

```
sayhello() # output is: Hello My Country
```

```
printmessage(msg) # output is: Hello My Country
```

```
print(msg) # output is: Hello My Country
```

هنگامی که از یک نام استفاده می کنیم ، مفسر پایتون ابتدا درونی ترین دامنه یعنی حوزه محلی را برای یافتن آن جستجو می کند ، اگر آن را در دامنه محلی نیابد نزدیک ترین دامنه ای که تعریف تابع را در بر گرفته است یعنی حوزه محلی تابع بیرونی آن را در صورت وجود جستجو خواهد کرد، اگر باز هم نیابد در حوزه سراسری فایل جاری و پس از آن در حوزه نام ذاتی جستجو را ادامه خواهد داد. چنانچه جستجو نتیجه ای در بر نداشته باشد مفسر پایتون خطای **Name Error** را اعلام خواهد کرد. به این ترتیب جستجوی یک نام در پایتون قانون **LEGB** گفته می شود که از سرواژه های **Local, Enclosing, Global, و Built-In** ساخته شده است مفهوم این قانون در شکل زیر به تصویر کشیده شده است :



تصویر ۲-۱۳ قانون LEGB

مثال :

```
def scope_test():
    def do_local():
        spam = "local spam"
    def do_nonlocal():
        nonlocal spam
        spam = "nonlocal spam"
    def do_global():
        global spam
        spam = "global spam"
    spam = "test spam"
    do_local()
    print("After local assignment:", spam)
    do_nonlocal()
    print("After nonlocal assignment:", spam)
    do_global()
    print("After global assignment:", spam)
scope_test()
print("In global scope:", spam)
```

خروجی کد بالا به شکل زیر است :

```
After local assignment: test spam
After nonlocal assignment: nonlocal spam
After global assignment: nonlocal spam
In global scope: global spam
```


فصل چهاردهم

شی گرایی در پایتون

هدف اصلی زبان های برنامه نویسی کمک به انسان ها برای حل مسائل دنیای واقعی است. گواه این گفته نقش بسیار پررنگ سامانه های نرم افزاری در چیرگی بر چالش های فراروی افراد و سازمان ها در محیط پیچیده و رقابتی کسب و کار های امروزی است. از این روطراحان زبان های برنامه نویسی با آفرینش رویکرد جدیدی به نام برنامه نویسی شی گرا^۱ دامنه مفاهیم، روش ها و ساختارهای موجود در زبان های برنامه نویسی را چنان گسترش داده اند که با استفاده از آنها به سادگی می توان مفاهیم و پدیده های دنیای واقعی را الگو سازی کرد و نمایش داد. آنان برای دست یابی به این خواسته مهم، یعنی نزدیک تر کردن مفاهیم دنیای برنامه نویسی با مفاهیم دنیای واقعی از طبیعت و جهان پیرامون ما الهام گرفته اند.

جهان هستی و محیط پیرامون ما از تعداد بی شماری پدیده های گوناگون شکل گرفته است که هریک دارای کارکرد و هستی مشخص و یگانه ای است اما می توانند با دیگر چیزها به شیوه های گوناگون و رنگارنگ ارتباط داشته باشد. و بر اساس ویژگی های مشترک با دیگر پدیده ها در یک گروه یا دسته بزرگتر طبقه بندی شود. برای نمونه در یک دسته قوی زیبا که به صورت گروهی و به گونه ای با شکوه پرواز می کنند هر یک از قوها دارای یک هویت یکتا و مستقل از دیگر قوهای گروه است. اما می تواند با دیگر قوها ارتباط داشته باشد. از دیدگاه طبقه بندی هر قو عضوی از تیره مرغابیان است که غازها و اردک ها را نیز در بر می گیرد، خود تیره مرغابیان در رده بزرگتری به نام پرندگان جای می گیرد و خود دسته پرندگان نیز در شاخه بزرگتری به نام مهره داران قرار دارد. و سرانجام مهره داران یک زیر مجموعه از طبقه بزرگتری به نام جانداران است.

تمامی پدیده ها و چیزهای موجود در دنیای واقعی صرف نظر از اینکه در چه طبقه ای قرار دارند دارای دو دسته ویژگی اصلی هستند: ویژگی های وصفی، ویژگی های رفتاری به زبان ساده تر

¹ Object - Oriented Programming - OOP

برخی از این ویژگی ها آن چیز یا پدیده را توصیف می کنند یعنی داده هایی در مورد آن به ما می دهند که می توان براساس آنها حالت یا وضعیت پدیده را مشخص کرد. برخی دیگر از ویژگی ها از جنس رفتار بوده و کار یا کارهایی را که یک پدیده می تواند انجام دهد بدست می دهند. برای نمونه چیزی مانند گوشی هوشمند را در نظر بگیرید ویژگی هایی چون رنگ گوشی ، وزن گوشی ، جنس بدنه ، نوع سیستم عامل ، مقدار حافظه داخلی و قیمت از جنس داده هستند و اطلاعاتی در باره آن بدست می دهند. اما ویژگی هایی چون دریافت و ارسال پیامک ، زنگ زدن ، زنگ خوردن ، عکس گرفتن و روشن / خاموش شدن از جنس رفتار بوده و بیانگر کاری است که یک گوشی هوشمند می تواند انجام دهد.

برنامه نویسی شی گرا

همانند دنیای واقعی که در بردارنده دو مفهوم هستی (چیز ، پدیده ، موجودیت) و طبقه بندی است. برنامه نویسی شی گرا^۱ نیز بر محور دو مفهوم اساسی کلاس^۲ و شی^۳ شکل گرفته است. هرچند که شی گرایی در برگیرنده مفاهیم و اصولی چون : کلاس ، شی، پوشیده سازی^۴ ، وراثت^۵ ، چند وجهی^۶ و انتزاع^۷ است اما ساختار اصلی و هسته مرکزی تشکیل دهنده این رویکرد برنامه نویسی مفهوم کلاس است. از این رو در این فصل بیشتر به این مفهوم پرداخته می شود.

¹ Object - Oriented Programming - OOP

² Class

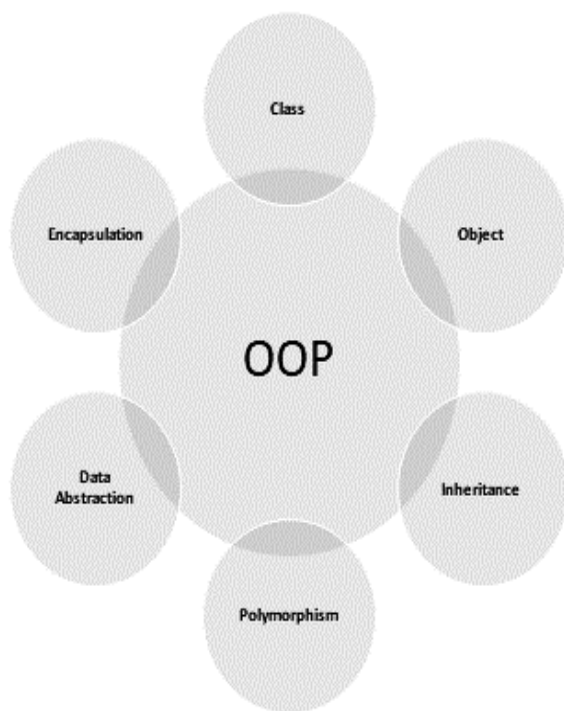
³ Object

⁴ Encapsulation

⁵ Inheritance

⁶ Polymorphism

⁷ Data Abstraction



شکل 1-14 : مفاهیم و اصول شی گرای

مفهوم کلاس

با یک نگاه مفهومی می توان گفت که کلاس در دنیای برنامه نویسی شی گرا همسان مفهوم طبقه بندی در جهان واقعی است و شی نیز برابر با همان هستی ، چیز یا پدیده های موجود در محیط پیرامون ماست. برای مثال افرادی که در صف دستگاه خودپرداز یک بانک پشت سر هم ایستاده اند. هر کدام یک هستی (چیز) جداگانه هستند که هویت مستقل و یکتای خود را دارند و می توانند در دسته بزرگتری به نام مشتریان بانک طبقه بندی شود ، مشتریان بانک هم می توانند در طبقه بزرگتری به نام انسان ها قرارگیرند. به زبان برنامه نویسی شی گرا هر مشتری بانک یک شی^۱ است که می تواند از روی کلاس^۲ مشتری نمونه سازی گردد به زبان ساده تر در زبان های شی گرا هر

^۱ object

^۲ class

شی از روی یک کلاس ساخته می شود به ساختن یک شی از یک کلاس نمونه سازی^۱ گفته می شود.

یک تفاوت ظریف بین پدیده های دنیای واقعی با مفهوم شی در برنامه نویسی شی گرا این است که در جهان پیرامون ما چیزها و پدیده ها وجود دارند و ما براساس ویژگی ها و رفتارهای مشترک می توانیم آنها را در طبقه مشخصی دسته بندی کنیم اما در دنیای برنامه نویسی شی گرا ابتدا کلاس مورد نظر ساخته می شود و سپس از روی آن یک یا بیش از یک شی مورد نیاز ساخته خواهد شد.

در برنامه نویسی شی گرا ، هر برنامه در قالب نهادهای کوچکی به نام شی که از روی کلاس ها ساخته می شوند و با یکدیگر برهم کنش دارند در نظر گرفته می شود. برای داشتن این اشیا ابتدا باید کلاس های مورد نیاز برنامه را تعریف کنیم. از این رو درک مفهوم کلاس در برنامه نویسی شی گرا از اهمیت بسیار بالایی برخوردار است. چرا که بدون وجود کلاس هیچ شی نمی تواند وجود خارجی داشته باشد. کلاس همانند نقشه یک ساختمان است این نقشه خود ساختمان نیست اما راهنمایی است که از روی آن یک خانه واقعی ساخته می شود. دردنیای زبان های برنامه نویسی شی گرا نیز همین گونه است ما ابتدا یک یا چندین کلاس تعریف می کنیم که همانند نقشه یک خانه در برگیرنده ویژگی ها و رفتارهایی است که در برنامه نیاز داریم ، سپس از روی این کلاس ها تعدادی شی می سازیم همانگونه که از روی یک نقشه می توان تعداد زیادی خانه ساخت از یک کلاس هم می توان به تعداد دلخواه شی ساخت به زبان ساده تر می توان گفت که کلاس یک نقشه ساخت است و شی نمونه ای است که بر اساس آن ساخته می شود.

هر کلاس رفتار^۲ و ویژگی های^۳ اشیا^۴ی که قرار است از آن ایجاد شوند را تعریف می کند. از یک کلاس به هر تعداد می توان شی ایجاد کرد. هر شی بیانگر یک نمونه از کلاس است به هر شی که از روی کلاس ساخته می شود یک نمونه^۴ از آن کلاس گفته می شود. پس از نمونه سازی ، شی دارای تمامی ویژگی ها و رفتارهایی است که در کلاس تعریف شده است. همانگونه که پیشتر گفته

¹ Instantiation

² Behavior

³ Attributes

⁴ instance

شد ، ساخت یک شی از یک کلاس نمونه سازی نامیده می شود. بر همین اساس دو نوع کلاس در زبان های شی گرا وجود دارد :

۱- کلاس های عادی که توانایی نمونه سازی دارند و به آنها کلاس های واقعی یا غیر انتزاعی^۱ گفته می شود.

۲- کلاس هایی که توانایی نمونه سازی ندارند و به آنها کلاس های انتزاعی^۲ گفته می شود.

هر کلاس باید توانایی پیاده سازی دو مفهوم رفتار^۳ و ویژگی^۴ را داشته باشد. ویژگی ها داده هایی هستند که به صورت یک یا چند متغیر در کلاس تعریف می شوند و می توانند هر شی ساخته شده از روی کلاس را توصیف کنند به زبان ساده تر داده هایی در مورد آن شی به ما می دهند. در فرهنگ واژگان شی گرایی به ویژگی های تعریف شده در یک کلاس اعضای داده ای^۵، فیلد^۶ یا صفت^۷ نیز گفته می شود.

رفتارها کار یا کارهایی که یک شی می تواند انجام دهد را تعریف می کنند. رفتار در یک کلاس به صورت تابع تعریف و پیاده سازی می شود از این رو به رفتارها، توابع عضو^۸ نیز گفته می شود. در شی گرایی به توابع تعریف شده در یک کلاس شگرد^۹ گفته می شود.

¹ concrete class

² Abstract Class

³ Behavior

⁴ Attribute

⁵ Data Members

⁶ Field

⁷ Attribute

⁸ Member Function

⁹ Method

تعریف کلاس در پایتون

در پایتون برای تعریف یک کلاس رهنمون `class` بکار گرفته می شود. الگوی زیر نشان دهنده چگونگی تعریف کلاس در پایتون است :

Class *ClassName*:

<statement 1>

.

.

.

<statement n>

همانگونه که دیده می شود نام کلاس پس از رهنمون `class` نوشته می شود و پس از آن نویسه دو نقطه بیانی (:): جای می گیرد . ناگفته نماند که پس از نام کلاس و پیش از نویسه : می توان یک جفت کمانک باز و بسته () قرار داد اما پیشنهاد می شود تنها زمانی از این نگارش استفاده کنید که بخواهید مفهوم وراثت را پیاده سازی کنید.

دستورات بدنه کلاس با اولین تورفتگی نسبت به رهنمون `class` آغاز و با آخرین تورفتگی نسبت به آن پایان می پذیرند. به زبان ساده تر هر سطر دستور موجود در یک کلاس با یک تورفتگی آغاز می شود. مقدار تورفتگی دلخواه است اما در سرتاسر بدنه کلاس یکنواختی در تورفتگی باید رعایت گردد. بیشتر برنامه نویسان مقدار تورفتگی را چهار فاصله خالی در نظر می گیرند.

بدنه یک کلاس نمی تواند خالی باشد از این رو در مواردی که به هردلیلی نمی خواهید دستوری در بدنه کلاس قرار گیرد می توانید از تنها یک دستور `pass` استفاده کنید.

مثال :

`class myclass:`

`pass`

مثال : کلاسی به نام مستطیل می سازیم که دارای دو صفت به نامهای طول (Length) و عرض (Width) است :

```
class Rectangle:
```

```
    length = 15
```

```
    width = 10
```

نمونه سازی

به عملیات ایجاد یک شی از یک کلاس نمونه سازی^۱ گفته می شود. از هر کلاس می توان بی شمار نمونه (شی) ساخت هر شی ساخته شده از یک کلاس حوزه^۲ ویژه خود را خواهد داشت که جدای از دیگر اشیای ساخته شده از همان کلاس خواهد بود از این رو نمونه های یک کلاس مستقل از یکدیگرند . به هنگام نمونه سازی از یک کلاس قراردادن جفت کمانک بازوبسته () پس از نام کلاس فراموش نشود. در کد زیر از روی کلاس Rectangle که در مثال قبل ساخته ایم سه شی (object) به نام های obj1 ، obj2 و obj3 می سازیم :

```
obj1 = Rectangle ()
```

```
obj2 = Rectangle ()
```

```
obj3 = Rectangle ()
```

```
print(obj1.width)
```

الگوی دسترسی به هر یک از اعضای یک شی همانند زیر است :

نام صفت یا متد . نام شی

مثال :

```
obj1.width
```

¹ Instantiation

² Scope

نکته مهم دیگری که لازم است به یاد داشته باشید این است که ساخت یک کلاس به معنای ایجاد یک نوع (type) جدید در برنامه است که می توان چندین شی یا نمونه از آن پدید آورد. نام کلاس بیانگر نوع^۱ نمونه ها یا همان اشیایی است که از روی آن ساخته می شوند. برای دست آوردن نوع یک شی از تابع type() استفاده می کنیم.

مثال :

```
class TestClass:
```

```
    pass
```

```
class Rectangle:
```

```
    pass
```

```
obj1 = TestClass()
```

```
rect1 = Rectangle()
```

```
print(type(obj1)) #output is : <class '__main__.TestClass'>
```

```
print(type(rect1)) # output is <class '__main__.Rectangle'>
```

کلاس ها نیز همانند توابع ، دامنه^۲ ویژه خود را دارند با تعریف هر کلاس یک حوزه محلی جدید در برنامه پایتون پدیدار می شود. به بیان دیگر هریک از اشیاء ساخته شده از یک کلاس دارای هویت^۳ مستقل از یکدیگر هستند و در مکانی جداگانه در حافظه قرار داده می شوند. تابع id() در پایتون برای نمایش شناسه یک شی بکاربرده می شود.

مثال :

```
class TestClass:
```

```
    pass
```

```
obj1 = TestClass()
```

```
obj2 = TestClass()
```

```
print(id(obj1)) # output is : 35888792
```

¹ type

² scope

³ identity


```
print(id(obj2)) # output is : 35888840
```

در زبان پایتون برای بررسی نوع^۱ یک شی می توانید از دو روش زیر استفاده کنید :

۱- بکار گیری تابع `type()`

۲- بکارگیری تابع `isinstance()`

تابع `isinstance()` دارای دو پارامتر است ، نخستین پارامتر شی مورد نظر است و دومین پارامتر نوع مورد نظر می باشد. چنانچه شی ارسال شده به تابع از نوع دریافت شده باشد ، تابع مقدار `True` و گرنه مقدار `False` را برمی گرداند.

مثال :

```
class Rectangle:
    pass
num = 255
rect1 = Rectangle()
print(isinstance(rect1,Rectangle))
print(isinstance(num,int))
```

سازنده یک کلاس

هنگام ساخت یک شی (ساخت یک نمونه جدید از کلاس) به صورت خودکارشگرد^۲ ویژه ای از درون کلاس فراخوانده می شود که به آن سازنده^۳ گفته می شود. فراخوانی خودکار این متد به برنامه نویس اجازه می دهد تا بتواند چگونگی ایجاد شی و مقداردهی اولیه به آن را اداره کند. در فرآیند نمونه سازی از یک کلاس ، پایتون دو متد ویژه زیر را صدا می زند :

۱- متد `__new__()`

۲- متد `__init__()`

¹ type

² Method

³ Constructor

وظیفه متد `__new__()` ساخت یک شی (نمونه) جدید از کلاس است و نخستین آرگومان ارسالی به آن نام کلاسی است که نمونه جدید باید از روی آن ساخته شود. به صورت خودکار و بی درنگ پس از اجرای متد `__new__()` و پیش از آنکه شی جدید ساخته شده از شگرد `__new__()` برگردانده شود متد ویژه دیگری به نام `__init__()` فراخوانی می شود. بنابراین نخستین پارامتر این متد شی جاری است یعنی همان شی که به دست متد `__new__()` ساخته شده است. نام پیش فرض نخستین پارامتر این متد `self` است. اما می توانید به دلخواه نام دیگری برای آن برگزینید اما همواره به خاطر داشته باشید که این پارامتر باید نخستین پارامتر تعریف شده در این شگرد باشد. آرگومان همسان این پارامتر که همان شی جاری است به وسیله مفسر پایتون به این شگرد ارسال می شود.

بر اساس آنچه که تاکنون از فرآیند نمونه سازی در پایتون می دانید می توان گفت که متد ویژه `__init__()` سازنده کلاس نیست و در فرآیند نمونه سازی در زبان پایتون دو متد ویژه `__new__` و `__init__` به همراه یکدیگر نقش سازنده کلاس را بازی می کنند. متد `__new__` شی را می سازد و متد `__init__` هم برای سفارشی کردن شی و دادن مقدار اولیه به آن بکار می رود. آرگومان های همسان با پارامترهای متد `__init__` به جز `self` که به وسیله مفسر پایتون مقداردهی می گردد باید در زمان نمونه سازی کلاس و به درستی به این متد ارسال شوند

هر کلاس پایتون دارای یک پیاده سازی پیش فرض از دو متد `__new__` و `__init__` است. بنابراین به هنگام ساخت یک کلاس ساده نیازی به پیاده سازی دوباره آنها وجود ندارد. در بسیاری از کلاس های کوچک و حتی بزرگ و پیچیده نیازی به پیاده سازی متد `__new__` احساس نمی شود اما پیاده سازی متد `__init__` در بیشتر کلاس ها ضرورتی گریزناپذیر است.

مثال :

```
class Rectangle:
    def __init__( self, Length, Width)
        self.Length = Length
        self. Width = Width
```

حال که کلاس دلخواه خود را به گونه ای ساخته ایم که دارای یک سازنده است به روش زیر می توانیم از روی آن به تعداد مورد نیاز شی (object) بسازیم :

```
obj1 = Rectangle(15,10)
print(obj1.width)
obj2 = Rectangle(26,17)
print(obj2.width)
```

مثال : در مثال زیر برای کلاس TestClass هردو متد __new__ و __init__ را پیاده سازی کرده ایم برنامه را اجرا و خروجی آن را مشاهده کنید :

```
class TestClass:
    def __new__(cls,*args,**kwargs):
        print("Call of __new__() Method ")
        print("class name : ",cls)
        print("Arbitrary Args : ",args)
        print("Arbitrary Key Word Args : " , kwargs)
        obj = super().__new__(cls)
        return obj
    def __init__(self, name, family, age):
        print("Call of __init__() Method ")
        self.name = name
        self.family = family
        self.age = age
test1 = TestClass("Omid","IranManesh",25)
```

صفت ها

در دنیای شی گرایی دو نوع صفت وجود دارند :

۱- صفت نمونه ای^۱ یا صفت وابسته به شی

¹ Instance Attribute

مقدار این نوع صفت ها از یک شی به شی دیگر متفاوت است. برای نمونه در مثال زیر مقدار هر یک از صفت های `name` ، `family` و `ident` در دو شی `prsn1` و `prsn2` که هر دو از روی یک کلاس به نام `Person` نمونه سازی شده اند می تواند متفاوت از یکدیگر باشد. به این گونه از صفت ها تنها با استفاده از نمونه (شی) می توان دسترسی داشت و با استفاده از نام کلاس قابل دستیابی نیستند از این رو در مثال زیر با افزودن دستور `print(Person.name)` به کد ، کامپایلر پایتون خطای `AttributeError: type object 'Person' has no attribute 'name'` را اعلام خواهد کرد.

مثال :

`class Person:`

`def __init__(self, ident , name, family):`

`self.ident = ident`

`self.name = name`

`self.family = family`

`prsn1 = Person(1020,"narges", "Iranmanesh")`

`prsn2 = Person(1030,"Ziba", "Bahary")`

`print(prsn1.name)`

`print(prsn2.name)`

۲- صفت کلاسی^۱

این صفت ها درون یک کلاس و بیرون از تمامی شگردهای آن تعریف می شوند. کاربرد این دسته از صفت ها به اشتراک گذاشتن یک متغیر در بین تمامی اشیای ساخته شده از روی کلاس است. به زبان ساده تر تمامی اشیای یک کلاس به `class Attribute` های آن دسترسی دارند. برای نمونه در مثال زیر مقدار هر یک از صفت های `uid` و `objcount` از نوع صفت های کلاس هستند.

مقدار این نوع صفت ها هم با استفاده از نام کلاس و هم با استفاده از هر یک از اشیای ساخته شده از روی کلاس قابل دستیابی است. برای نمونه در مثال زیر می توان نوشت :

¹ Class Attribute

```
print(Person.objcount)
```

```
print(prsn2.objcount)
```

برای تغییر مقدار یک صفت کلاس در بیرون از کلاس ، از نام خود کلاس استفاده می‌شود. و در درون کلاس با استفاده از شگردهای کلاس^۱ صورت می‌پذیرد. برای نمونه در مثال زیر می‌توان نوشت :

```
Person.uid = "Bc45HKM"
```

نکته بسیار مهمی که باید همواره به خاطر داشته باشید این است که برای تغییر مقدار یک صفت کلاس هرگز از شی استفاده نکنید چرا که این کار موجب ساخته شدن یک صفت شی^۲ جدید برای آن شی شده و تغییری در مقدار صفت کلاس ایجاد نخواهد کرد. همانگونه که در مثال زیر دیده می‌شود برای افزودن صفت به یک شی از متد `__init__` استفاده می‌شود. و این یکی از کاربردهای اصلی متد `__init__` در پایتون است.

مثال :

```
class Person:
```

```
    uid = "AB10RTGH"
```

```
    objcount = 0
```

```
    def __init__(self, ident , name, family):
```

```
        self.ident = ident
```

```
        self.name = name
```

```
        self.family = family
```

```
    def __new__(cls, *args, **kwargs):
```

```
        cls.objcount += 1
```

```
        obj = super().__new__(cls)
```

```
        return obj
```

¹ Class Methods

² Instance Attribute

```

prsn1 = Person(1020,"narges", "Iranmanesh")
print(Person.objcount)
print(prsn1.objcount)
prsn2 = Person(1030,"Ziba", "Bahary")
print(Person.objcount)
print(prsn2.objcount)
print(Person.uid)
Person.uid = "Bc45HKM"
print(prsn1.uid)
print(Person.uid)
print(prsn2.uid)

```

شگردهای یک کلاس

شگردها^۱ همان توابع تعریف شده در یک کلاس هستند که رفتار یا کارهایی که یک شی ساخته شده از روی کلاس می تواند انجام دهد را تعیین می کنند. در دنیای شی گرایی در پایتون شگردها بر سه دسته اند :

۱- شگردهای وابسته به شی^۲

شگردهای شی پرکاربردترین نوع شگرد در پایتون هستند. این شگردها تنها با استفاده از نام شی^۳ قابل دسترس هستند و نمی توان با استفاده از نام کلاس به آنها دسترسی داشت. به زبان ساده تر این شگردها در کلاس تعریف می شوند اما برای دسترسی به آنها ناگزیر باید یک شی از روی کلاس ساخته شود و سپس با استفاده از نام شی ، شگرد مورد نظر فراخوانی گردد. الگوی فراخوانی این شگردها همانند زیر است :

ObjectName.InstanceMethodName()

¹ Methods

² Instance Methods

³ object

برای نمونه در مثال زیر شی rect1 از روی کلاس Rectangle ساخته می شود و در این کلاس شگرد شی به نام calc_area() پیاده سازی شده است از این رو فراخوانی این متد به شکل زیر خواهد بود :

```
rect1. calc_area()
```

نام پیش فرض نخستین پارامتر این نوع شگردها self است. می توانید به دلخواه خود نام دیگری برای آن برگزینید اما همواره به خاطر داشته باشید که این پارامتر باید نخستین پارامتر تعریف شده در شگرد باشد. آرگومان همسان این پارامتر که همان شی جاری است به وسیله مفسر پایتون به این دسته از شگردها ارسال می شود و نیازی به ارسال آن از سوی برنامه نویس نیست.

مثال :

```
class Person:
```

```
def Instance_method(self):
```

```
    pass
```

مثال : برای کلاس مستطیل (Rectangle) دو متد به نامهای محاسبه مساحت و محاسبه محیط تعریف می کنیم :

```
class Rectangle:
```

```
    def __init__(self,Length, Width):
```

```
        self.Length = Length
```

```
        self. Width = Width
```

```
    def calc_area(self) :
```

```
        return self.Length * self.Width
```

```
    def calc_circum(self):
```

```
        return 2*( self.Length * self.Width)
```

```
rect1 = Rectangle(22, 14)
```

```
print(rect1. calc_area())
```

```
print(rect1. calc_circum())
```

۲- شگردهای کلاس^۱

این دسته از شگردها با رهنمون `@classmethod` به کامپایلر معرفی می شوند. در این دسته از شگردها همواره باید دست کم یک پارامتر تعریف شود. نام پیش فرض نخستین پارامتر این متد `cls` است. می توانید به دلخواه نام دیگری برای آن برگزینید اما همواره به خاطر داشته باشید که این پارامتر باید نخستین پارامتر تعریف شده در این شگرد باشد. آرگومان همسان این پارامتر که همان **کلاس جاری** است به وسیله مفسر پایتون به این دسته از متدها ارسال می شود و نیازی به ارسال آن از سوی برنامه نویس نیست. این دسته از شگردها از اشیای ساخته شده از یک کلاس چیزی نمی دانند و تنها کلاس را می شناسد بنابراین تنها می توانند `Class Attribute` ها را دستیابی و دستکاری کنند. این دسته از متدها را می توان هم با استفاده از نام کلاس و هم با استفاده از اشیای ساخته شده از روی کلاس فراخوانی کرد.

```
class Person:
```

```
    @classmethod
    def class_method(cls):
        pass
```

مثال :

```
class Student:
    univercity = "Sharif"
    def __init__(self,stdid,name, family):
        self.stdid = stdid
        self.name = name
        self.family = family

    @classmethod
    def set_name(cls,univercity_name) :
```

¹ Class Methods


```
cls.university = university_name
```

```
std1 = Student(2014,"nazanin", "kashany")
print(Student.university)
print(std1.university)
Student.set_name("tehran")
std2 = Student(20115,"fereshteh", "kashany")
print(Student.university)
print(std1.university)
std2.set_name("Shiraz University")
print(Student.university)
print(std2.university)
print(std1.university)
```

3- شگردهای ایستا^۱

یک شگرد ایستا نمی تواند به اعضای کلاس یا نمونه (شی) دسترسی داشته باشد چرا که نخستین پارامتر آن `self` یا `cls` نیست. به بیان دیگر این شگردها تنها با داده هایی که به صورت آرگومان به آنها ارسال می شود کار می کنند. این شگردها هم با استفاده از نام کلاس و هم با استفاده از نام شی قابل دستیابی هستند. این دسته از شگردها با استفاده از رهنمون `@staticmethod` ساخته می شوند.

مثال :

```
class Calculator:
    @staticmethod
    def Add(x, y) :
        return x + y
print(Calculator.Add(5,7))
calc2 = Calculator()
print(calc2.Add(5,7))
```

¹ Static Methods

مثال :

```
class Person:
```

```
    @staticmethod
```

```
    def static_method():
```

```
        pass
```

مثال :

```
class Employee:
```

```
    def __init__(self,name, salary, project_name):
```

```
        self.name = name
```

```
        self.salary = salary
```

```
        self.project_name = project_name
```

```
    def work(self):
```

```
        task = self.gather_requirements(self.project_name)
```

```
        print("Completed :",task)
```

```
    @staticmethod
```

```
    def gather_requirements(project_name):
```

```
        if project_name == "ABC":
```

```
            reuirement = "task1"
```

```
        else:
```

```
            reuirement = "task2"
```

```
        return reuirement
```

```
emp1 = Employee("Amir",5000000,"HTC")
```

```
emp1.work()
```

دستکاری اشیاء

در زبان برنامه نویسی پایتون می توان ویژگیهای یک شی را ویرایش کرد، حذف کرد و یا ویژگی های جدیدی به آن افزود. برای حذف یک ویژگی در شی از دستور `del` استفاده می شود. همچنین می توانید با بکارگیری دستور `del` یک شی را به طور کامل حذف کنید

مثال : ویرایش ویژگی های یک شی ساخته شده از روی کلاس `Rectangle` (این کلاس در مثال های پیشین تعریف شده است)

```
rect2 = Rectangle(22,14)
```

```
rect2. Length = 36
```

```
rect2. Width = 22
```

```
print(rect2. calc_area())
```

مثال : حذف ویژگی یک شی

```
del rect2. Length
```

مثال : حذف یک شی

```
del rect2
```

مثال : ساخت ویژگی جدید برای شی

```
class test:
```

```
    pass
```

```
test1 = test()
```

```
test1.name = 'Monitor'
```

```
test1.price =255
```

```
print(test1.price)
```

ارث بری در پایتون

همانگونه که در جهان واقعی مفهومی به نام ارث بری وجود دارد و هر فرزند می تواند ویژگی ها و رفتارهایی چون رنگ چشم ، رنگ پوست ، حالت کلی چهره و دیگر صفات را از پدر و مادر خود به ارث ببرد. در شی گرای نیز این مفهوم وجود دارد. یعنی می توان از یک کلاس موجود کلاس یا کلاس های دیگری ایجاد کرد که صفت ها و شگردهای کلاس قبلی را به ارث ببرند. برای فرزند یک کلاس می توان شگردها و صفات جدید تعریف کرد و یا می توان شگردهای کلاس پدر را بازنویسی کرد. کلاسی که می خواهیم کلاس های دیگر آن را به ارث ببرند کلاس پایه^۱ یا ابر کلاس^۲ نامیده می شود و کلاس یا کلاس هایی که از کلاس پایه ارث می برند را کلاس فرزند، زیر کلاس^۳ و یا کلاس مشتق شده^۴ می نامند.

در زبان پایتون نام کلاس پایه در کمانک بازویسته پس از نام کلاس فرزند قرار می گیرد ، چنانچه کلاس فرزند از بیش از یک کلاس پایه ارث بری کند کلاس های پایه با نویسه , از هم جدا می شوند :

```
class BaseClass1:
```

```
    pass
```

```
class BaseClass2:
```

```
    pass
```

```
class BaseClass3:
```

```
    pass
```

```
class ChildClass(BaseClass1, BaseClass2, BaseClass3):
```

```
    pass
```

برای آشنایی با روش پیاده سازی مفهوم وراثت در پایتون ابتدا کلاسی به نام **persons** می سازیم که قرار است به عنوان کلاس پدر برای کلاسی به نام **employee** نقش بازی کند از این رو ابتدا

¹Base Class

² Super Class

³ Sub class

⁴ Derived class

کلاس پدر را می سازیم این کلاس دارای سه ویژگی به نام های : `firstname` , `lastname` و `personid` است که به ترتیب نام ، نام خانوادگی و شناسه ملی فرد را نمایندگی می کنند همچنین این کلاس دارای شگردی به نام `getfullname` است که نام کامل فرد را برمی گرداند.

class persons:

```
def __init__(self, name , family,identify):
```

```
    self.firstname = name
```

```
    self.lastname = family
```

```
    self.personid = identify
```

```
def getfullname(self):
```

```
    return self.personid + ":" + self.firstname + " " + self.lastname
```

```
prsn1 = persons("Arman" , "Izanlou" , "0650041235")
```

```
print(prsn1.getfullname())
```

حال نوبت به تعریف کلاس فرزند فرا رسیده است، کلاس فرزند دلخواه ما `employee` نام دارد و دارای ۷ ویژگی به نام های `department` , `salary` , `personid` , `lastname` , `firstname` و `emploeetype` است که به ترتیب نام ، نام خانوادگی ، شناسه ملی ، میزان حقوق ، بخش محل کار و نوع استخدام کارمند را مشخص می کنند. همچنین کلاس مورد نظر ما دارای شگردی به نام `get_net_salary()` است که خالص حقوق دریافتی کارمند را محاسبه می کند:

```

class employee(persons):
    def __init__(self,name,family,identify,salary,department,emplooeetype):
        self.firstname = name
        self.lastname = family
        self.personid = identify
        self.salary = salary
        self.department = department
        self.emplooeetype = emplooeetype

    def get_net_salary(self, taxrate):
        return self.salary - ((taxrate * self.salary)/ 100)

emp = employee("Arash","Izanlou", "097571438",1900000000,"it",
               "Rasmi")
print(emp.getfullname())
print (emp.get_net_salary(15))

```

هنگامی که متد `__init__()` را به کلاس فرزند اضافه می کنید کلاس فرزند دیگر متد `__init__()` کلاس پدر را به ارث نخواهد برد به بیان ساده تر سازنده کلاس فرزند سازنده کلاس پدر را لغو کرده و آن را نادیده می گیرد. از این رو چنانچه به هر دلیلی کلاس فرزند نیاز به ارث بری از متد سازنده کلاس پدر داشته باشد یعنی در کنار سازنده ویژه خود نیاز به سازنده کلاس پدر نیز داشته می توانید به یکی از دو شیوه زیر این کار را انجام دهید :

۱- فراخوانی متد سازنده کلاس پدر در بدنه متد سازنده کلاس فرزند با استفاده از نام کلاس پدر

این روش به ویژه زمانی مفید است که کلاس فرزند از بیش از یک کلاس پایه ارث بری کند هنگامی که از این شیوه بهره می گیرید به یاد داشته باشید که نخستین پارامتر متد `__init__()` کلاس پدر یعنی `self` هم باید در فهرست آرگومان های ارسالی به آن وجود داشته باشد.

مثال :

```
persons.__init__(self,name , family,identify)
```

۲- فراخوانی متد سازنده کلاس پدر در بدنه متد سازنده کلاس فرزند با استفاده از رهنمون `super()`

هنگامی که از این شیوه بهره می گیرید به یاد داشته باشید که نخستین پارامتر متد `(__init__)` کلاس پدر یعنی `self` نباید در فهرست آرگومان های ارسالی به آن وجود داشته باشد. وگرنه مفسر پایتون اعلام خطا خواهد کرد.

مثال :

```
super().__init__(name , family,identify)
```

برای بیان بهتر آنچه که گفته شد مثال بالا را با روش های گفته شده بازنویسی می کنیم:

```
class persons:
```

```
    def __init__(self, name , family,identify):
```

```
        self.firstname = name
```

```
        self.lastname = family
```

```
        self.personid = identify
```

```
    def getfullname(self):
```

```
        return self.personid + ":" + self.firstname + " " + self.lastname
```

```
prsn1 = persons("Arman" , "Izanlou" , "0650041235")
```

```
print(prsn1.getfullname())
```

```
class employee(persons):
```

```
    def __init__(self, name,family,identify,salary,department  
,emplooeetype):
```

```
        super().__init__(name , family,identify)
```

```
        # persons.__init__(self,name , family,identify)
```

```

self.salary = salary
self.department = department
self.employeetype = employeetype

def get_net_salary(self, taxrate):

    return self.salary - ((taxrate * self.salary)/ 100)

emp = employee("Arash" , "Izanlou" , "097571438", 1900000000 , "it" ,
"Rasmi")
print(emp.getfullname())
print (emp.get_net_salary(15))

```

ارث بری از چند کلاس

در پایتون یک کلاس فرزند می تواند از چند کلاس پایه ارث بری کند یعنی همزمان می تواند شگردها و صفات چندین کلاس را به ارث ببرد. برای این منظور لازم است نام کلاس های پایه بعد از نام کلاس فرزند و درون یک جفت کمانک باز و بسته آورده شود هر کلاس پایه با یک نویسه , از کلاس دیگر جدا می شود.

مثال : در این مثال کلاس Student از کلاس های Person و Human ارث بری می کند

Class Student(Person,Human):

pass

مثال : کلاس دانشجوی دوره کارشناسی (bstudents) می تواند از دو کلاس افراد (persons) و کلاس دانشجویان (students) ارث بری کند با کلاس person پیشتر آشنا شدیم کلاس student دارای ویژگی های دوره تحصیلی (course) ، دانشگاه (university) و رشته تحصیلی (study) است و نیز دارای شگردهای به نام get_course_info() است. افزون بر این کلاس فرزند (bstudents) دارای دو ویژگی دانشکده (college) و سال تحصیلی (academicyear) است.


```
class persons:
    def __init__(self, name , family,identify):
        self.firstname = name
        self.lastname = family
        self.personid = identify
    def getfullname(self):
        return self.personid + ":" + self.firstname + " " + self.lastname
```

```
class students:
    def __init__(self,course, study,univercity):
        self.course = course
        self.study = study
        self.univercity = univercity

    def get_course_info(self):
        return str(self.course) + " " + self.study + " " + self.univercity
```

```
class bstudent(persons, students):

    def __init__(self, name , family, identify, course, study,univercity,
academicyear, college):

        persons.__init__(self, name , family,identify)
        students.__init__(self,course, study,univercity)
        self.academicyear = academicyear
        self.college = college

bstd1 = bstudent("Arash", "izanlou", "0860781587", 4, "Computer
Engineer", "Sharif" ,1400, "Fanni")
print(bstd1.get_course_info())
```

زمانی که یک کلاس فرزند از چندین کلاس پایه ارث بری می کند و در کلاس فرزند نیاز به فراخوانی شگرد سازنده کلاس های پایه باشد **مهم ترین چالش** پیش رو ترتیب دستیابی به هریک از شگردهای `__init__` موجود در کلاس های پایه است. برای مثال اگر در تمامی کلاس های پایه شگرد `__init__` پیاده سازی شده باشد با توجه به اینکه این متد در تمامی آنها دارای نامی یکسان است مفسر پایتون چگونه می تواند آرگومان های ارسال شده به شگرد `__init__` کلاس فرزند را از بین تمام کلاس های پایه به شگرد `__init__` کلاس پایه مورد نظر ارسال کند. همانگونه که پیشتر گفته شد ساده ترین شیوه چیره شدن بر این چالش بکارگیری نام کلاس پایه است در این روش برای هر پارامتر تعریف شده در شگرد `__init__` کلاس های پایه باید آرگومان همسان با آن به شگرد `__init__` کلاس فرزند ارسال شود. از جمله برای پارامتر `self` که همواره نخستین پارامتر است.

مثال :

```
class BaseClass1:
```

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

```
class BaseClass2:
```

```
    def __init__(self, family):
        self.family = family
```

```
class BaseClass3:
```

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

```
class ChildClass(BaseClass1, BaseClass2, BaseClass3):
```

```
    def __init__(self, name, family, age ,salary):
        self.salary = salary
        BaseClass1.__init__(self, name)
        BaseClass2.__init__(self, family)
        BaseClass3.__init__(self, age)
```

```
    def printinfo(self):
```

```
        print(self.name + " " + self.family + "\n" + str(self.age) + "\n" +
str(self.salary))
```

```
child1 = ChildClass("Arash", "Izanlou", 25, 253000000)
child1.printinfo()
```

روش دوم، بکارگیری تابع `super()` است، از این تابع در کلاس فرزند برای دستیابی به هر یک از اعضای کلاس پایه استفاده می شود. این تابع برای پیمایش کلاس های پایه که کلاس فرزند از آنها ارث بری می کند و یافتن متد مورد نظر از الگوریتمی به نام 'MRO' بهره می برد. هرکلاس پایتون یک صفت ویژه به نام `__mro__` دارد که دربردارنده ترتیب کلاس هایی است که پایتون براساس آن به دنبال یک متد می گردد. این ترتیب نتیجه بدست آمده از الگوریتم MRO است. برای نمونه در مثال بالا دستور `print(ChildClass.__mro__)` نتیجه زیر را برمیگرداند :

```
(<class '__main__.ChildClass'>, <class '__main__.BaseClass1'>, <class '__main__.BaseClass2'>, <class '__main__.BaseClass3'>, <class 'object'>)
```

همانگونه که در نتیجه بدست آمده از دستور دیده می شود در این مثال مفسر پایتون برای جستجوی یک پارامتر در شگرد مورد نظر (در مثال ما شگرد `__init__`) ابتدا داخل خود کلاس `ChildClass` را بررسی می کند و سپس شروع به پیمایش کلاس های پایه آن به ترتیب `BaseClass1`، `BaseClass2` و `BaseClass3` خواهد کرد. همواره آخرین کلاسی که پیمایش خواهد شد کلاس `object` است. این کلاس ابرکلاسی است که تمامی کلاس های پایتون به صورت ضمنی از آن ارث بری می کنند.

با آگاهی از نتیجه الگوریتم MRO که همانا ترتیب پیمایش کلاس های پایه است متد مورد نظر را به هنگام فراخوانی تابع `super()` مقداردهی خواهیم کرد. یعنی ترتیب تعریف پارامترها و نیز ترتیب ارسال آرگومان ها به این شگرد در کلاس فرزند را به گونه ای انجام می دهیم که گویی در ابتدا قرار است شگرد همسان آن در کلاس پایه مقداردهی گردد.

نکته بسیار مهمی که همواره باید به خاطر داشته باشید این است که مفسر پایتون با بدست آوردن نخستین نتیجه موفق در یافتن شگرد مورد نظر پایش را پایان می دهد از این رو نیاز است تا هریک از شگردهای همسان با شگرد مورد نظر ما در کلاس های پایه نیز تابع `super()` را فراخوانی کنند.

¹ Method Resolution Order

همچنین لازم است تا شگردهای همسان در کلاس های پایه بگونه ای تعریف شوند که بتوانند هر تعداد پارامتر را بپذیرند از این رو آخرین پارامتر این متدها ***args** خواهد بود. این پارامتر تمامی آرگومان های اضافی ارسال شده به متد را در خود نگه می دارد. از این رو برای ادامه روند فراخوانی متدهای همسان باقی مانده تنها کافیست که این مقدار ارسال گردد.

مثال :

```
class BaseClass1:
```

```
    def __init__(self, name, *args):
```

```
        self.name = name
```

```
        super().__init__(*args)
```

```
class BaseClass2:
```

```
    def __init__(self, family,*args):
```

```
        self.family = family
```

```
        super().__init__(*args)
```

```
class BaseClass3:
```

```
    def __init__(self, age,*args):
```

```
        self.age = age
```

```
        super().__init__(*args)
```

```
class ChildClass(BaseClass1, BaseClass2, BaseClass3):
```

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

```
        self.salary = salary
```

```
        super().__init__(name, family, age)
```

```
    def printinfo(self):
```

```
        print(self.name + " " + self.family + "\n" + str(self.age) + "\n" +
```

```
str(self.salary))
```

```
child1 = ChildClass("Arash", "Izanlou",25, 253000000)
```

```
child1.printinfo()
```

فصل پانزدهم

مدیریت خطا

رخ دادن خطا به هنگام اجرای یک برنامه به دلایل مختلفی چون تقسیم بر صفر، ورودی های نادرست، و یا پدید آمدن شرایطی چون کمبود حافظه، قطعی شبکه و خرابی سخت افزار پدید می آید و می تواند مسیر عادی جریان برنامه را از حالت طبیعی و پیش بینی شده خارج و سبب توقف ناخواسته برنامه گردد. رخ دادن خطا رویدادی است که در بسیاری از برنامه های نوشته شده ممکن است رخ دهد، در حقیقت وجود خطا همزاد همیشگی نرم افزار است و گریزی از آن نیست. هیچ برنامه نویسی نمی تواند از رخ ندادن خطا در کدی که می نویسد اطمینان پیدا کند اما با استفاده از ساختارهای مناسبی که در زبان های برنامه نویسی برای مدیریت و رسیدگی به خطا پیش بینی شده است می تواند خطای احتمالی رخ داده در برنامه را مدیریت کند و اطمینان یابد که خطا به حال خود رها نشده و به طور شایسته ای به آن رسیدگی می شود.

در فرهنگ واژگان زبان های برنامه نویسی، به خطا یا مشکلی که به هنگام اجرای برنامه رخ می دهد و جریان عادی برنامه را از مسیر طبیعی خود خارج می سازد و موجب توقف اجرای برنامه و پایان یافتن غیر عادی آن می شود یک استثنا^۱ گفته می شود. یک استثنا اگر به درستی رسیدگی نشود می تواند اجرای برنامه را با شکست روبرو سازد. از این رو در زبان های برنامه نویسی مدیریت استثنا بخش بسیار مهمی از زبان است. مدیریت استثنا سازوکاری برای برای به دام انداختن و رسیدگی به خطای رخ داده در طول اجرای برنامه است. این سازوکار در زبان برنامه نویسی پایتون با استفاده از ساختاری به نام `try` انجام می شود. الگوی کلی نگارش و بکارگیری این ساختار به شکل زیر است :

¹ Exception

try:

try Block

except:

except Block

else:

else Block

finally:

Finally Block

هر بخش جداگانه از این ساختار دارای نقشی متفاوت در فرآیند به دام انداختن و رسیدگی به خطاست :

بخش **try** محل به دام انداختن خطای رخ داده است. پس قطعه^۱ کدی از برنامه که ممکن است موجب رخ دادن یک استثنا شود، در بدنه رهنمون **try** نوشته می شود. وجود بخش **try** در ساختار مدیریت خطا اجباری است.

بخش **except** مسئول رسیدگی به استثنای رخ داده است به بیانی ساده تر قطعه کدی که خطای رخ داده در بخش **try** را مدیریت می کند در بدنه رهنمون **except** نوشته می شود، کد نوشته شده در بدنه این رهنمون تنها در صورتی اجرا می شود که به هنگام اجرای برنامه، در کد نوشته شده در بدنه رهنمون **try** خطایی به دام افتاده باشد. وجود بخش **except** در مدیریت خطا اجباری است. ساختار مدیریت خطا می تواند دارای بیش از یک بخش **except** باشد.

اگر می خواهید در صورت رخ ندادن خطا در بخش **try** کد خاصی اجرا شود آن را در بدنه رهنمون **else** بنویسید. قطعه کد نوشته شده در بخش **else** تنها زمانی اجرا می شود که در قطعه کد نوشته شده در بخش **try** استثنایی رخ نداده باشد. وجود بخش **else** در مدیریت خطا اجباری نیست. و می توان از نوشتن آن چشم پوشی کرد.

بخش **finally** مسئول اجرای کدی است که باید همواره اجرا شود یعنی قطعه کد نوشته شده در بدنه رهنمون **finally** بدون توجه به اینکه در کد نوشته شده در بدنه **try** خطایی رخ داده است

¹ Block

یا نه ، اجرا خواهد شد. وجود بخش `finally` در مدیریت خطا اجباری نیست و اگر به آن نیاز نباشد می توان از نوشتن آن چشم پوشی کرد. بسیاری از برنامه نویسان از این بخش برای آزاد سازی منابع مهمی چون اتصالات پایگاه داده ، بستن ارتباطات شبکه و بستن فایل هایی که در طول اجرای برنامه با استفاده از رهنمون `open` باز می شوند استفاده می کنند.

مثال : در برنامه زیر با رسیدن اجرای برنامه به عبارت `n = divided / divisor` خطای تقسیم بر صفر رخ می دهد. بنابراین اجرای برنامه متوقف شده و مفسر پایتون خطای `ZeroDivisionError: division by zero` را اعلام خواهد کرد.

```
divided = 45
divisor = 0
n = divided / divisor
print(n)
divisor = int(input("Please Enter Number: "))
n = divided / divisor
print(n)
```

مثال : باز نویسی برنامه مثال قبل با استفاده از ساختار مدیریت استثنا ، در کد باز نویسی شده بارخ دادن خطا در قطعه `Try` برنامه در هم نمی شکند و متوقف نمی شود بلکه با نمایش پیام مناسب به کاربر اجرای برنامه را از نخستین خط پس از قطعه `except` از سر می گیرد.

```
divided = 45
divisor = 0
try:
    n = divided / divisor
    print(n)
except:
    print("Error : Divided by Zero")
divisor = int(input("Please Enter Number: "))
n = divided / divisor
print(n)
```

به دام انداختن نوع خاصی از استثنا

رویکرد درست در پاسخ به استثنای رخ داده در برنامه این است که نوع استثنایی که می خواهیم در بخش `try` به دام انداخته شود را مشخص کنیم با این کار می توان به هر استثنا بر اساس نوع و ماهیت آن پاسخ مناسبی داد و از رسیدگی به تمامی خطاهای رخ داده با روشی یکسان دوری گزید. به این منظور می توانید نوع استثنای مورد نظر خود را در بخش `except` معرفی کنید. در ساختار مدیریت خطا امکان استفاده از چند بخش `except` متفاوت که هر یک استثنای خاصی را رسیدگی می کند نیز وجود دارد.

مثال : در کد زیر فقط استثنای `ZeroDivisionError` به دام می افتد این استثنا زمانی رخ می دهد که عملوند سمت راست عملگر تقسیم عدد صفر باشد. یعنی تقسیم بر صفر رخ دهد.

```
divided = ۴۵
```

```
divisor = ۰
```

```
try:
```

```
    n = divided /divisor
```

```
    print(n)
```

```
except ZeroDivisionError as exp:
```

```
    print(exp)
```

مثال : در کد زیر هم استثنای `ZeroDivisionError` و هم استثنای `NameError` به دام می افتند. استثنای `NameError` زمانی رخ می دهد که مفسر پایتون در دامنه محلی و یا سراسری نتواند یک نام بکاررفته در کد را بیابد. در مثال زیر متغیر `x` در دستور `print(x)` استفاده شده در حالی که در هیچ دامنه ای از کد تعریف نشده است.

```
divided = ۴۵
```

```
divisor = ۱۵
```

```
try:
```

```
    n = divided /divisor
```

```
    print(n)
```

```
    print(x)
```

```
except ZeroDivisionError as exp:
```



```
print(exp)
except NameError as exp۲:
    print(exp۲)
```

مثال : استثنای `ValueError` زمانی رخ می دهد که یک مقدار نادرست به یک عبارت یا یک تابع ارسال گردد برای مثال ورودی تابع یا متغیر به کار رفته در عبارت از نوع عددی است اما مقداری از نوع رشته ای به آن داده می شود . برای نمونه در کد زیر تابع `int()` مقدار گرفته شده از کاربر توسط دستور `input()` را به یک عدد صحیح تبدیل می کند اما چنانچه مقدار گرفته شده از کاربر غیر قابل تبدیل شدن به یک عدد صحیح باشد (برای مثال کاربر مقدار "aa" را وارد کند) استثنای `ValueError` رخ خواهد داد.

```
while True:
    try:
        x = int(input("Pleas Enter Numbers:"))
        break
    except ValueError:
        print("That was not valid number, Try again")
```

می توانید چندین نوع استثنا را در قالب یک `tuple` و تنها با یک رهنمون `except` رسیدگی کنید:

مثال :

```
divided = 45
divisor = 15
try:
    n = divided /divisor
    print(n)
    print(x)
except (ZeroDivisionError,NameError, TypeError) as exp:
    print(exp)
```

به هنگام استفاده از ساختار برنامه نویسی `try` به دام انداختن و رسیدگی به خطا به شکل زیر انجام می شود :

- ۱- نخست قطعه کد نوشته شده در بدنه `try` اجرا می شود.
- ۲- اگر استثنایی در بخش `try` رخ ندهد ، بخش `except` نادیده گرفته می شود و بخش `else` یا `finally` در صورت وجود اجرا خواهند شد ، فرآیند مدیریت خطا پایان می یابد و اجرای برنامه از نخستین خط کد پس از ساختار `try` ادامه خواهد یافت.
- ۳- اگر استثنایی در بخش `try` به دام انداخته شود یعنی در قطعه کد موجود در بدنه `try` استثنایی رخ دهد اجرای کد در همان نقطه ای که خطا رخ داده است پایان می یابد و از اجرای باقی مانده کد چشم پوشی می شود. حال چنانچه نوع استثنای رخ داده با نوع استثنای تعریف شده در هریک از `except` ها مطابقت داشته باشد قطعه کد موجود در بدنه `except` وابسته اجرا می شود و پس از آن بخش `finally` در صورت وجود اجرا خواهد شد و سپس اجرای برنامه از نخستین خط کد پس از ساختار `try` ادامه می یابد.
- ۴- اگر نوع استثنای رخ داده با هیچ یک از استثنای تعریف شده در `except` ها مطابقت نداشت. مفسر پایتون استثنای رخ داده را یک استثنای مدیریت نشده^۱ شناسایی می کند و با نمایش یک پیغام خطا به اجرای برنامه پایان می دهد.

مثال : در کد زیر نوع استثنایی که رخ می دهد تقسیم بر صفر یا همان `ZeroDivisionError` است و چون نوع استثنای رخ داده برابر با هیچ یک از انواع استثنای تعریف شده در `except` ها نیست مفسر پایتون خطای رخ داده را به عنوان یک استثنای مدیریت نشده شناسایی می کند و با نمایش پیغام خطای `division by zero` به اجرای برنامه پایان می دهد.

```
divided = 45
divisor = 0
try:
```

¹ unhandled exception

```
n = divided /divisor
print(n)
except NameError as exp2:
    print(exp2)
except OverflowError as exp3:
    print(exp3)
print("hello World")
```

دستور raise

برای وادار کردن مفسر پایتون به اعلان یک استثنا می توانید از رهنمون **raise** استفاده کنید برای مثال فرض کنید تابعی برای تبدیل گاهشمار جلالی (هجری خورشیدی) به گاهشمار میلادی نوشته اید ، این تابع دارای سه ورودی به نامهای **JalaliYear** ، **JalaliMonth** و **JalaliDay** است که به ترتیب سال ، ماه و روز تاریخ جلالی را در قالب سه عدد صحیح به عنوان ورودی می پذیرد و چون در تقویم جلالی هر سال به ۱۲ ماه تقسیم می شود بنابراین اگر آرگومان ارسالی به تابع برای پارامتر **JalaliMonth** عددی کوچکتر از ۱ یا بزرگتر از ۱۲ باشد لازم است با تولید یک استثنای مناسب کد فراخواننده تابع را از این موضوع آگاه کنید تا بتواند خطای رخ داده را به درستی مدیریت کند.

مثال :

```
def jalali2Milady(JalaliYear, JalaliMonth , JalaliDay ):
    if JalaliMonth < 1 or JalaliMonth >12:
        raise Exception("Error Wrong Month value")
    print("{}\{}\{}".format(JalaliYear , JalaliMonth , JalaliDay) )
try:
    jalali2Milady(1402, 4, 21)
    jalali2Milady(1402, 14, 21)
except Exception as exp:
    print(exp)
```

مثال :

```
def self_sum_int(number):
```

```

if not isinstance(number, int):
    raise TypeError("Wrong Type of input number")
return number + number

try:
    print(self_sum_int(55))
    print(self_sum_int('aa'))
except Exception as exp:
    print(exp)

```

ایجاد یک استثنای سفارشی

در زبان برنامه نویسی پایتون با ساخت کلاسی که از کلاس Exception یا یکی از زیرکلاسهای آن ارث می برد می توان یک Exception سفارشی ساخت.

مثال :

```

class JalaliCalenderException(Exception):
    def __init__(self, number, message):
        self.number = number
        self.message = message
        super().__init__(self.message)
    def __str__(self):
        return f'JalaliCalenderException:{self.number} {self.message}'

def jalali2Milady(JalaliYear, JalaliMonth , JalaliDay ):
    if JalaliMonth < 1 or JalaliMonth >12:
        raise JalaliCalenderException(JalaliMonth, "Error:Wrong
Month value")
    return "{}\{}\{}".format(JalaliYear , JalaliMonth , JalaliDay)

try:
    print(jalali2Milady(1402,14,21))
except JalaliCalenderException as je:
    print(je)

```

فصل شانزدهم

فایل ها

متغیرها و ساختمان داده های مختلفی چون فهرست^۱، واژه نامه^۲، چندتایی^۳، کلاس و دیگر اشیایی که تا کنون در برنامه ها جهت ذخیره و پردازش اطلاعات استفاده می کردیم در حافظه اصلی^۴ رایانه ذخیره می شوند بنابراین با پایان یافتن برنامه یا خاموش شدن دستگاه همه داده ها از بین می روند. برای ذخیره دائمی، داده ها باید در فایل ها ذخیره شوند. از این رو کار با فایل ها بخش بسیار مهمی از هر برنامه است. فایل یک واحد منطقی در سیستم عامل^۵ است که برای نگهداری طولانی مدت اطلاعات بر روی رسانه های ذخیره سازی مختلفی چون دیسک سخت^۶، دیسک های نوری^۷ و دیگر رسانه های ویژه نگهداری طولانی مدت اطلاعات استفاده می شود. کار با فایل ها فرآیندی بسیار پیچیده است از این رو مدیریت فایل ها یکی از وظایف بسیار مهم سیستم عامل است.

فایلها در دو گروه متنی^۸ و دودویی^۹ دسته بندی می شوند، در فایل های متنی، داده ها به صورت رشته (نوع داده str) ذخیره می شوند و به صورت رشته هم خوانده می شوند. به زبان ساده تر واحد خواندن و نوشتن داده ها در فایل های متنی رشته است به همین دلیل است که اطلاعات ذخیره شده در فایل های متنی (برای مثال یک فایل html) قابل چاپ کردن و دیده شدن توسط انسان هستند یعنی کاربر می تواند اطلاعات ذخیره شده در یک فایل متنی را به کمک نرم افزار ساده ای چون

^۱ list

^۲ dictionary

^۳ tuple

^۴ RAM

^۵ Operating System (OS)

^۶ Hard Disk

^۷ CD , DVD

^۸ Text

^۹ Binary

Notepad در سیستم عامل ویندوز و نرم افزارهای Vim یا Nano در سیستم عامل لینوکس مشاهده کند.

در فایل های متنی ، داده ها در قالب نوع داده ای رشته^۱ ذخیره و خوانده می شوند از این رو نوع رمزگذاری^۲ متن استفاده شده در این فایلها برای ذخیره صحیح ، خواندن درست و نمایش مناسب داده های ذخیره شده در آن بسیار مهم است، رمزگذاری یونیکد utf-8 یکی از پرستفاده ترین رمزگذاری های مورد استفاده در فایل های متنی است. برای دیدن نوع رمزگذاری پیش فرض سیستم عامل می توانید از کد زیراستفاده کنید :

```
import locale
print(locale.getpreferredencoding(False))
```

در فایلهای دودویی^۳ داده ها به صورت مجموعه ای از بایت ها (نوع داده bytes) ذخیره می شوند و به صورت بایت هم خوانده می شوند به زبان ساده تر واحد خواندن و نوشتن داده ها در فایل های دودویی بایت است. یعنی داده ها به صورت مجموعه ای از بیت های ۰ و ۱ و در قالب آرایه ای از بایت ها در فایل ذخیره می شوند و به همان صورت هم خوانده می شوند.فایلهای تصویری (برای مثال JPG و BMP)، فایل های صوتی (برای مثال MP3 و WAV) ، فایلهای ویدئویی (برای مثال MPEG، DIVX) وفایل های اجرایی (برای مثال EXE و COM) نمونه هایی از فایلهای دودویی هستند. در زبان برنامه نویسی پایتون برای کار کردن با فایل ها لازم است سه گام اساسی زیربه ترتیب برداشته شوند:

۱- بازکردن فایل با استفاده از تابع open()

۲- انجام عملیات مورد نظر (خواندن ، نوشتن ، به روزرسانی و ...)

۳- بستن فایل

¹ string

² encoding

³ Binary

تابع open()

گل سرسید توابع و دستورات کار با فایل ها در پایتون دستور `open()` است از این دستور برای باز کردن یک فایل استفاده می شود. باز کردن یک فایل به معنای درخواست مفسر پایتون از سیستم عامل برای در اختیار گرفتن آن فایل و اجازه یافتن برای اجرای عملیات مختلفی چون خواندن از فایل یا نوشتن در فایل است اگر تلاش دستور `open` برای باز کردن فایل با موفقیت همراه شود یک شی فایل^۱ برگردانده می شود که از آن می توان برای کار با فایل باز شده استفاده کرد چنانچه تلاش این دستور برای باز کردن فایل با شکست روبرو شود یک استثنا از نوع `OsError` رخ می دهد.

الگوی کلی نگارش و بکار گیری این دستور همانند زیر است :

`open(path , mode)`

تابع `open` دارای چندین ورودی ظاهری^۲ است که در زیر به دو تا از مهم ترین آنها پرداخته شده است:

path : این ورودی مسیر دستیابی به فایل شامل نام و مکان ذخیره سازی فایل را مشخص می کند.

mode : این ورودی نوع عملیات (خواندن ، نوشتن و...) و حالت باز شدن فایل (متنی یا دودویی) را مشخص می کند.

نوع عملیات ، کاری است که می خواهید پس از باز شدن فایل با آن انجام دهید ، عملیات اصلی به هنگام کار با فایل ها عبارتند از : خواندن از فایل ، نوشتن در فایل ، ساخت فایل جدید ، به روزرسانی محتوای فایل و افزودن محتوا به انتهای فایل.

مقدار پیش فرض ورودی `mode`، باز شدن فایل برای عملیات خواندن در حالت متنی است یعنی اگر از مقداردهی به این ورودی چشم پوشی کنید دستور `open` نوع عملیات را خواندن و حالت فایل را متنی فرض می کند.

^۱ File Object

^۲ Parameter

مقدارهای مجاز قابل تخصیص به نوع عملیات و حالت در زیر آمده است :

۱- مقدارهای تعیین کننده نوع عملیات :

r یعنی باز کردن فایل برای عملیات خواندن ، این مقدار پیش فرض برای نوع عملیات است.

w یعنی بازکردن فایل برای عملیات نوشتن، این مقدار محتوای پیشین فایل را ازبین می برد. چنانچه فایل مورد نظراز قبل وجود نداشته باشد دستور `open` آن را خواهد ساخت.

x یعنی باز کردن فایل برای عملیات ساختن فایل جدید و عملیات نوشتن ، چنانچه فایل مورد نظراز قبل وجود داشته باشد یک استثنا رخ می دهد.

a یعنی باز کردن فایل برای عملیات افزودن محتوا به انتهای فایل ، چنانچه پیش ازاین فایل مورد نظرووجود نداشته باشد دستور `open` در ابتدا آن را می سازد.

$r+$ یعنی بازکردن فایل برای به روزرسانی وانجام همزمان عملیات خواندن و نوشتن، دراین حالت داده های قبلی فایل ازبین نمی روند.

$w+$ یعنی بازکردن فایل برای به روزرسانی وانجام همزمان عملیات خواندن و نوشتن، دراین حالت داده های قبلی فایل ازبین می روند.

$a+$ یعنی بازکردن فایل برای به روزرسانی وانجام همزمان عملیات خواندن و افزودن محتوا به انتهای فایل ، دراین حالت داده های قبلی فایل ازبین نمی روند. چنانچه پیش ازاین فایل مورد نظرووجود نداشته باشد دستور `open` آن را خواهد ساخت.

۲- مقدارهای تعیین کننده حالت فایل :

t یعنی باز کردن فایل در حالت متنی ، این مقدار پیش فرض برای حالت بازکردن فایل است

b یعنی بازکردن فایل در حالت دودویی

خواندن فایل

گفتیم که اگر تلاش دستور `open` برای بازکردن یک فایل با موفقیت همراه شود یک شی فایل^۱ برگردانده می شود که از آن می توان برای انجام عملیات مورد نیاز بروی فایل استفاده کرد. در زیر به مهمترین شگردهای این شی برای خواندن داده ها از فایل می پردازیم :

۱- شگرد `read()`

۲- شگرد `readline()`

۳- شگرد `readlines()`

شگرد `read()` ویژه خواندن از فایلی است که برای عملیات خواندن باز شده است این شگرد تنها دارای یک ورودی است که تعداد بایت ها (در حالت دودویی) یا رشته هایی (در حالت متنی) که در هر نوبت فراخوانی آن باید از فایل خوانده شوند را تعیین می کند. چنانچه مقدار این ورودی خالی رها شود و یا یک مقدار منفی به آن ارسال گردد، شگرد `read()` تمام محتوای فایل را می خواند و برمی گرداند.

خروجی این شگرد بستگی به حالت باز شدن فایل دارد ، در حالت متنی خروجی آن رشته ای است به طول n که همان آرگومان ارسال شده به شگرد است و در حالت دودویی خروجی آن دنباله ای است از بایت ها به طول n که همان آرگومان ارسال شده به شگرد است.

با هر بار فراخوانی این شگرد به صورت `read(n)` تعداد^۲ نویسه^۲ یا بایت^۳ (بر اساس حالت باز شدن فایل) خوانده شده و برگشت داده می شود با رسیدن به پایان فایل ، شگرد مقدار رشته خالی ("در حالت متنی و "b" در حالت دودویی) را برمی گرداند.

بستن فایل

بازکردن یک فایل با استفاده از دستور `open` به معنای درخواست مفسر پایتون از سیستم عامل برای در اختیار گرفتن آن فایل و اجازه یافتن برای اجرای عملیات مختلفی چون خواندن از فایل

^۱ File Object

^۲ character

^۳ Byte

یا نوشتن در فایل است در طول مدتی که مفسر پایتون فایل باز شده را در دست دارد دیگر پردازش ها نمی توانند به آن دسترسی داشته باشند. فایل ها از منابع محدود و حیاتی سیستم عامل هستند بنابراین بستن فایل یا فایل هایی که به وسیله دستور `open()` باز شده اند به معنای آزاد شدن منابع سیستم عامل و باز شدن دست آن برای اختصاص آنها به پردازش های دیگر است. از این رو بستن فایل های باز شده توسط دستور `open` پس از پایان عملیات مورد نیاز یک رویکرد بسیار خوب برای هر برنامه نویسی است. در پایتون برای بستن فایل باز شده توسط دستور `open` از شگردهای `close()` شی فایل برگشت داده شده استفاده می شود.

مثال: کد زیر فایل `country.txt` که در مسیر `d:\data\country.txt` قرار دارد را برای عملیات خواندن و در حالت متنی بازمی کند (در این فایل فقط رشته "IRAN" ذخیره شده است) سپس با استفاده از دستور `read(1)` نویسه به نویسه محتوای آنرا خوانده و چاپ می کند. سرانجام با استفاده از شگردهای `close()` فایل باز شده را می بندد:

```
country = open("d:\\data\\country.txt","tr")
print(country.read(۱))
print(country.read(۱))
print(country.read(۱))
print(country.read(۱))
country.close()
```

خروجی کد بالا همانند زیر است:

```
I
R
A
N
```

مثال: کد مثال بالا با توجه به این که شگردهای `read()` با رسیدن به پایان فایل، رشته خالی (") را برمی گرداند باز نویسی شده است:

```
country = open("d:\\data\\country.txt","tr")
```

```
char = country.read(۱)
```

```
while char != ":
```

```
    print(char)
```

```
    char = country.read(۱)
```

```
country.close()
```

مثال: کد زیر فایل student.txt که در مسیر d:\data\student.txt قرار دارد را برای عملیات خواندن و در حالت متنی باز می کند، شی فایل برگشت داده شده را در متغیر myfile ذخیره می کند. سپس با استفاده از شگردها read() تمامی محتوای فایل student.txt را خوانده و در خروجی صفحه نمایش چاپ می کند و سرانجام با استفاده از شگردها close() فایل باز شده را می بندد:

```
myfile = open("d:\\data\\student.txt", "tr")
```

```
print(myfile.read())
```

```
myfile.close()
```

با توجه حالت پیش فرض ورودی mode که بازکردن فایل برای عملیات خواندن در حالت متنی است می توان از ارسال مقدار "tr" به این آرگومان چشم پوشی کرد:

```
myfile = open("d:\\data\\student.txt")
```

```
print(myfile.read())
```

```
myfile.close()
```

چنانچه فایل مورد نظر در پوشه جاری یعنی در کنار اسکریپت پایتون ذخیره شده باشد نیازی به نوشتن مسیر کامل دسترسی به فایل نیست و ارسال تنها نام فایل به آرگومان path کافی است:

```
myfile = open("student.txt", "tr")
```

```
print(myfile.read())
```

```
myfile.close()
```

خواندن خط به خط فایل های متنی

براساس آنچه گفته شد یک فایل متنی را می توان مجموعه ای از رشته ها دانست که در قالب یک یا چند خط سازماندهی شده اند. بنابراین در فایل های متنی باید راهی برای تشخیص پایان یک خط وجود داشته باشد به این منظور سیستم عامل انتهای یک خط را با نشانه ویژه ای علامت گذاری می کند که به آن علامت پایان خط^۱ گفته می شود. برای نمونه سیستم عامل ویندوز پایان یک خط را با علامت CRLF^۲ که همان جفت نویسه "\r\n" است علامت گذاری می کند در حالی که سیستم عامل لینوکس پایان یک خط را تنها با نویسه LF^۳ که همان نویسه آشنای "\n" است علامت می زند. در زبان برنامه نویسی پایتون برای خواندن خط به خط یک فایل متنی سه روش وجود دارد که در زیر به هریک از آنها پرداخته می شود :

۱- استفاده از حلقه for

مثال :

```
myfile = open("d:\\data\\student.txt","r")
```

```
for line in myfile:
```

```
    print(line)
```

```
myfile.close()
```

۲- استفاده از شگرد readline()

شگرد readline() یک خط از فایل را می خواند و در قالب یک رشته (str) برمی گرداند سپس اشاره گر فایل را به ابتدای خط بعدی می برد. این شگرد با رسیدن به انتهای فایل متنی رشته خالی

¹ End Of Line (EOL)

² Carriage Return Line Feed

³ Line Feed

”) را برمی گرداند. اگر می خواهید تعداد معینی از رشته های موجود در یک خط از فایل را بخوانید تعداد مورد نظر خود را به عنوان آرگومان به این شگرد ارسال کنید.

مثال:

```
myfile = open("d:\\data\\student.txt", "r")
line= myfile.readline()
while line != "":
    print(line)
    line= myfile.readline()
myfile.close()
```

۳- استفاده از شگرد `readlines()`

شگرد `readlines()` تمامی خطوط موجود در فایل متنی را به یکباره خوانده و در قالب یک `list` برمی گرداند. بنابراین می توان با پیمایش فهرست برگردانده شده به هر خط از فایل دست یافت.

مثال:

```
myfile = open("d:\\data\\student.txt", "r")
list1 = myfile.readlines()
for itm in list1:
    print(itm)
myfile.close()
```

نوشتن در فایل

اگر تلاش دستور `open` برای بازکردن یک فایل با موفقیت همراه شود یک شی فایل^۱ برگردانده می شود که از آن می توان برای انجام عملیات مورد نیاز بر روی فایل استفاده کرد. در زیر به مهمترین شگردهای این شی برای نوشتن داده ها در فایل می پردازیم :

۱- شگرد `write()`

۲- شگرد `writelines()`

مهم ترین شگرد این شی برای نوشتن داده ها در فایل `write()` است. این شگرد ویژه نوشتن در فایلی است که برای عملیات نوشتن (`w`)، افزودن به انتهای فایل (`a`)، ساختن (`x`) یا به روزرسانی (+) باز شده است و تنها دارای یک ورودی است که بایت ها (در حالت دودویی) یا رشته ای (در حالت متنی) که باید در فایل نوشته شود را تعیین می کند.

مثال : کد زیر فایل `student.txt` را برای عملیات افزودن محتوا به انتهای فایل و در حالت متنی باز می کند ، با استفاده از شگرد `write()` دو خط متن به انتهای فایل اضافه کرده و فایل را می بندد سپس دوباره آنرا برای خواندن باز کرده و محتوای آنرا بر روی صفحه نمایش می دهد:

```
std = open("d:\\data\\student.txt","a")
std.write("zohre, bahary, ۲۰, ۹, Shirvan" + "\n")
std.write("nasim, kamai, ۱۹, ۱۲, Tehran" + "\n")
std.close()
std = open("d:\\data\\student.txt","r")
print(std.read())
std.close()
```

¹ File Object

مثال : کد زیر فایل متنی جدیدی به نام `teachers.txt` و در مسیر `d:\data` می سازد با استفاده از شگردهای `write()` دو خط متن به فایل اضافه کرده و فایل را می بندد سپس دوباره آنرا برای خواندن باز کرده و محتوای آنرا بر روی صفحه نمایش می دهد:

```
std = open("d:\\data\\teachers.txt","x")

std.write("zohre, bahary, ۲۰, ۹,Shirvan" + "\n")

std.write("nasim, kamai, ۱۹, ۱۲,Tehran" + "\n")

std.close()

std = open("d:\\data\\teachers.txt","r")

print(std.read())

std.close()
```

مثال : شبیه سازی دستور `copy` سیستم عامل ویندوز با نوشتن تابعی به نام `pycopy` که دارای دو ورودی است، ورودی نخست به نام `src` مشخص کننده نام و مسیر فایل مبدا است ، ورودی دوم به نام `dest` مشخص کننده نام و مسیری است که می خواهید فایل مبدا با آن نام و در آن مسیر ذخیره گردد:

```
def pycopy(src, dest):
    srcfile = open(src,"br")
    dstfile = open(dest,"bw")
    buff = ۲۰۴۸
    readedByte = srcfile.read(buff)
    while readedByte != b'':
        dstfile.write(readedByte)
        readedByte = srcfile.read(buff)
    print("file copied")
pycopy("d:\\۱\\img۱.jpg", "d:\\۲\\img۲.jpg")
```

بستن خودکار فایل ها

گفتیم که بازکردن یک فایل با استفاده از دستور `open` به معنای درخواست مفسر پایتون از سیستم عامل برای در اختیار گرفتن آن فایل و اجازه یافتن برای اجرای عملیات مختلفی چون خواندن از فایل یا نوشتن در فایل است در طول مدتی که مفسر پایتون فایل باز شده را در دست دارد دیگر پردازش ها نمی توانند به آن دسترسی داشته باشند. فایل ها از منابع محدود و حیاتی سیستم عامل هستند. بنابراین بستن فایل یا فایل هایی که به وسیله دستور `open()` باز شده اند به معنای آزاد شدن منابع سیستم عامل و باز شدن دست آن برای اختصاص آنها به پردازش های دیگر است. از این رو بستن فایل های باز شده توسط دستور `open` پس از پایان عملیات مورد نیاز یک رویکرد بسیار خوب برای هر برنامه نویسی است. اما این تنها دلیل استفاده از دستور `close()` نیست در حقیقت پایتون با هدف افزایش کارایی و بهبود سرعت نوشتن داده ها در فایل به صورت پیش فرض از یک حافظه میانی^۱ استفاده می کند یعنی به جای نوشتن مستقیم داده ها در دیسک سخت ، ابتدا آنها را به صورت موقت در بخشی از حافظه اصلی جای می دهد و سپس از روی این حافظه موقت داده ها را برداشته و در فایل می نویسد. حال چنانچه در بافر داده هایی وجود داشته باشد و برنامه نویس نوشتن دستور `close()` را فراموش کند ممکن است بخشی از داده های منتقل نشده از بافر به دیسک برای همیشه از دست بروند. در حقیقت مفسر پایتون با رسیدن به دستور `close()` ابتدا داده های باقی مانده در حافظه موقت میانه را که هنوز در فایل نوشته نشده اند در فایل می نویسد و سپس فایل را می بندد و این دلیل مهم دیگری برای بستن فایلها پس از انجام عملیات مورد نیاز است. همچنین گاهی خطا و استثنای رخ داده در برنامه موجب عبور جریان عادی هدایت برنامه از کد مربوط به بستن فایل می شود. در این حالت دستور `close()` هرگز اجرا نخواهد شد برای مثال در کد زیر در صورت رخ دادن خطا در خط سوم که شگرد `write()` فراخوانی شده است دستور `myfile.close()` اجرا نمی شود :

```
myfile = open("d:\\data\\student.txt","w")
myfile.write("Hellow World")
myfile.close()
```

¹ Buffer

بنا بر آنچه گفته شد ، وجود یک رویکرد مناسب برای چیره شدن بر این چالش ها و اطمینان از بسته شدن فایل ضرورتی گریزناپذیر است. در زبان برنامه نویسی پایتون دو شیوه برای حل این مشکل وجود دارد : ۱- بکارگیری رهنمون `try` ۲- بکارگیری رهنمون `with`

بکارگیری رهنمون `try`

رخ دادن خطا به هنگام کار با فایل ها به دلایل مختلفی چون: خرابی دیسک ، پر شدن فضای دیسک ، حذف فایل ، قفل شدن فایل توسط یک پردازش دیگر، ورود نام و مسیر نادرست برای دسترسی به فایل ویا پدید آمدن شرایطی چون : خرابی سخت افزار پدید می آید و می تواند مسیر عادی جریان برنامه را از حالت طبیعی و پیش بینی شده خارج و سبب توقف ناخواسته اجرای برنامه وعبور ناخواسته از کد نوشته شده برای بستن درست فایل های باز شده گردد. از این رو استفاده از ساختار `try` برای مدیریت و رسیدگی به خطای پیش آمده ضروری است. درحقیقت با قرار دادن دستور `close()` در بخش `finally` این ساختار می توان از بسته شدن فایل در صورت رخ دادن خطا اطمینان حاصل کرد برای آشنایی بیشتر با این رویکرد و درک بهتر آن به دو مثال زیر توجه کنید :

مثال : در کد زیر اگر در خط نخست که از دستور `open()` استفاده شده است استثنایی رخ ندهد و تلاش دستور `open()` برای باز کردن فایل `product.txt` با موفقیت همراه باشد حتی در صورت رخ دادن خطا به هنگام اجرای دستور `write()` فایل به درستی بسته خواهد شد. زیرا دستور `close()` نوشته شده در بدنه رهنمون `finally` در هر حال اجرا خواهد شد.

```
products = open("d:\\data\\product.txt","w")
```

```
try:
```

```
    products.write("Leyboard,۲۰۲")
```

```
except:
```

```
    print("Error: can not write to file" )
```

```
finally:
```

```
    products.close()
```

مثال : در کد زیر از ساختار `try` تو در تو استفاده شده است بنابراین افزون بر آنکه احتمال رخ دادن استثنا در دستور `write()` اداره می شود استثنای رخ داده در دستور `open()` هم پوشش داده خواهد شد.

try:

```
products = open("d:\\data۲\\product.txt", "w")
```

try:

```
products.write("Leyboard,۲۰۲")
```

except:

```
print("Error: can not write to file" )
```

finally:

```
products.close()
```

except:

```
print("Error: can not open file")
```

در مثال بالا به این نکته ظریف توجه کنید که اگر از ساختار `try` به صورت تو در تو استفاده نمی کردیم دستور `close()` را همانند کد زیر در بخش `finally` ساختار `try` تک سطحی می نوشتیم با رخ دادن خطای احتمالی در دستور `open()` جریان اجرای برنامه سرانجام وارد بخش `finally` می شد و با اجرای دستور `close()` سعی در بستن فایلی می کرد که پیش از این باز نشده است و این خود موجب اعلام یک استثنای مدیریت نشده توسط مفسر پایتون و توقف برنامه می گردید :

try:

```
products = open("d:\\data۲\\product.txt", "w")
```

```
products.write("Leyboard,۲۰۲")
```

except:

```
print("There is an Error")
```

finally:

```
products.close()
```

بکارگیری رهنمون with

هر منبع خارجی مانند فایل ، اتصال شبکه یا اتصال به پایگاه داده که تلاش برای بازکردن آن با استفاده از رهنمون with اداره شود با پایان یافتن بدنه رهنمون with به صورت خودکار بسته خواهد شد. در مقایسه با رویکرد بکارگیری ساختار try با بهره گیری از رهنمون with می توانید کد تمیزتر و ایمن تری بنویسید. الگوی کلی نگارش و بکارگیری این رهنمون همانند زیر است :

with expression as target_var:

do_something

مثال :

with open("d:\\data\\product.txt","w") as products:

products.write("Leyboard,۲۰۲")

توجه کنید که کد بالا تنها بسته شدن فایل در صورت موفق بودن عملیات باز شدن فایل را تضمین می کند و هر خطایی که ممکن است در جریان تلاش برای باز کردن فایل رخ دهد را نمی تواند مدیریت کند و این گونه خطاها به ناگزیر باید با استفاده از ساختار try اداره شوند درمثال زیر برای اطمینان از بسته شدن فایل و اداره استثناهای رخ داده در دستور open() ترکیبی از دو رویکرد try و with بکار گرفته شده است:

مثال :

try :

with open("d:\\data\\product.txt","w") as products:

products.write("Leyboard,۲۰۲")

except FileNotFoundError as e:

print(e)

پایان کتاب