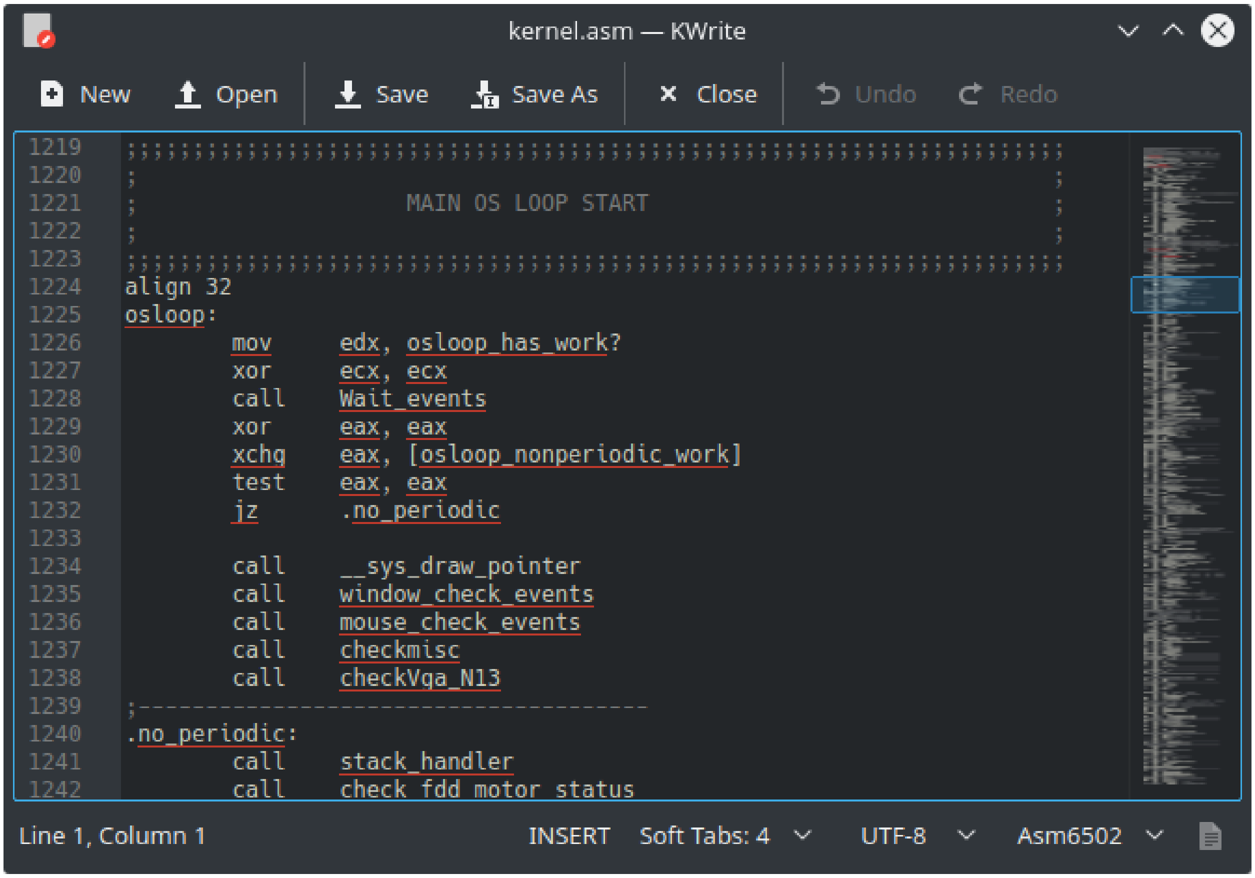
آیا سرعت زبان های برنامه نویسی باهم متفاوت است؟ چرا؟

زبان های برنامه نویسی زیادی وجود دارند که هریک دارای ویژگی ها و خصوصیات خاص خود هستند. باتوجه به موارد استفاده و نوع طراحی، بعضی از این برنامه ها نسبت به بقیه سرعت بیشتری دارند. البته این سرعت بیشتر گاهی هزینه های خاص خود را دارد. تفاوت های این زبان ها باعث تفاوت در کارایی آن ها می شود.

زبان برنامه نویسی چیست؟

علی رغم تفاوت های گسترده زبان های برنامه نویسی، کاربرد یکسان ان ها واداشتن کامپیوتر به انجام کارها است. تمام کدهای نوشته شده در نهایت به یک سری از اعداد مبهم ترجمه می شوند. می توان گفت که تمام زبان های برنامه نویسی (از جمله زبان اسمبلی که اعداد را به کلمات قابل خواندن تبدیل می کند) برای ساده تر کردن تولید نرم افزارها طراحی شده اند.



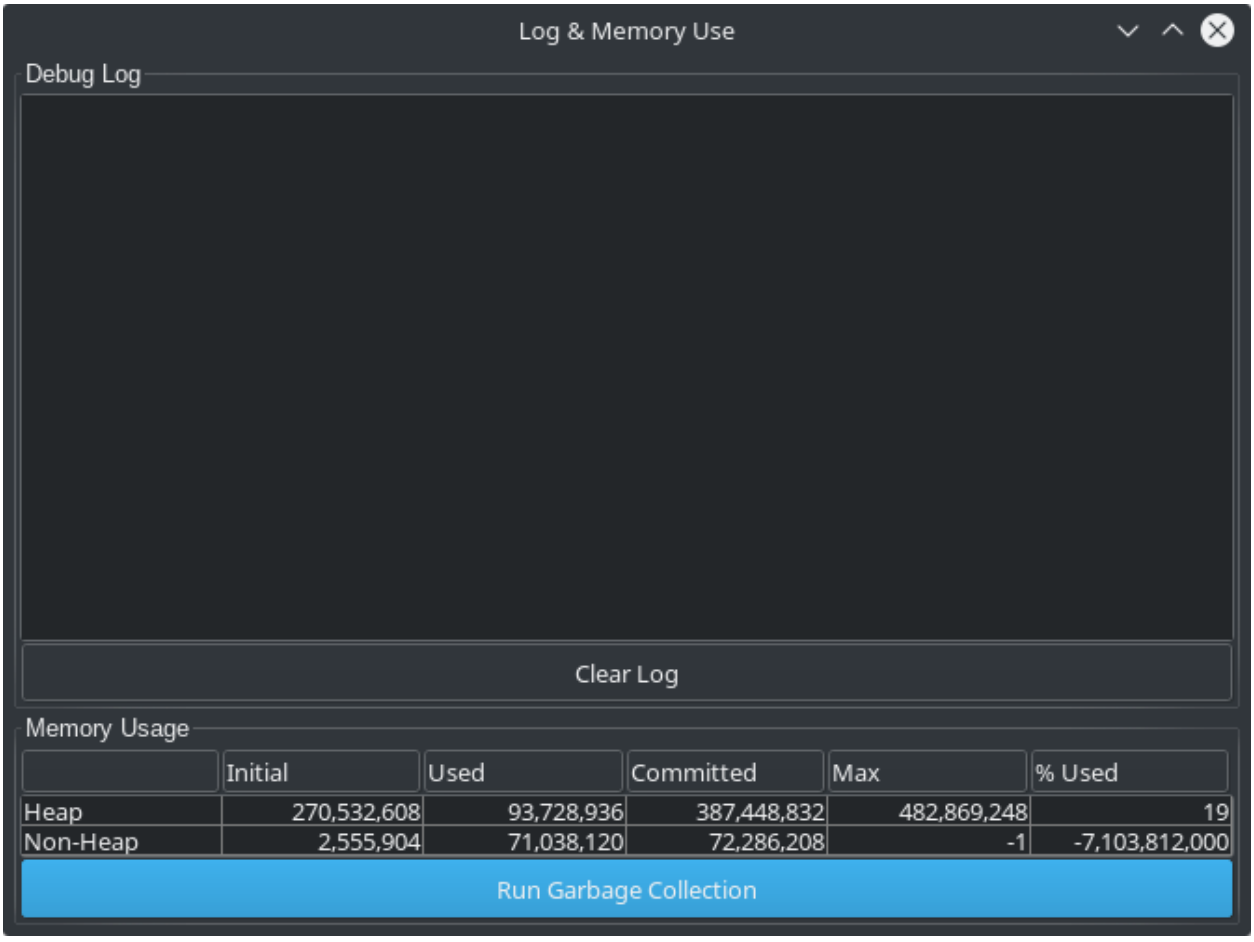
این زبان ها را می توان بر طبق سطح انتزاع آن ها طبقه بندی کرد. سطح انتزاع به معنی میزان انجام اموری است که در سطوح پایین تر باید بصورت دستی انجام شوند. هرچه سطح انتزاع، بالاتر باشد سطح زبان برنامه نویسی بالاتر بوده و برنامه نویسی ساده ترمی شود چون برنامه نویس به یادیگری و به خاطر سپردن موارد کمتری نیاز دارد.

یکی از این موراد مدیریت حافظه است. در برنامه های سطح پایین تر برنامه نویس باید بصورت دستی مقدار رم لازم برای اجرای برنامه را تعیین کرده و پس از پایان کار آن مقدار را ازاد کند. اگر برنامه نویس رم را ازاد نکند یا اتفاقی رخ دهد که برنامه نویس آن را در نظر نگرفته نباشد میزان استفاده از رم بیشتر و بیشتر می شود. زبان های سطح بالاتر مثل جاوا اینکار را بصورت خودکار انجام می دهند.

هزینه اتوماسیون

علی رغم مزیت های فراوان زبان های سطح بالا، استفاده از آن ها جنبه های منفی را نیز به همراه دارد. برای مثال هرچه میزان کنترل برنامه نویس بر سرعت کاهش یابد، کارایی برنامه نیز کمتر می شود.

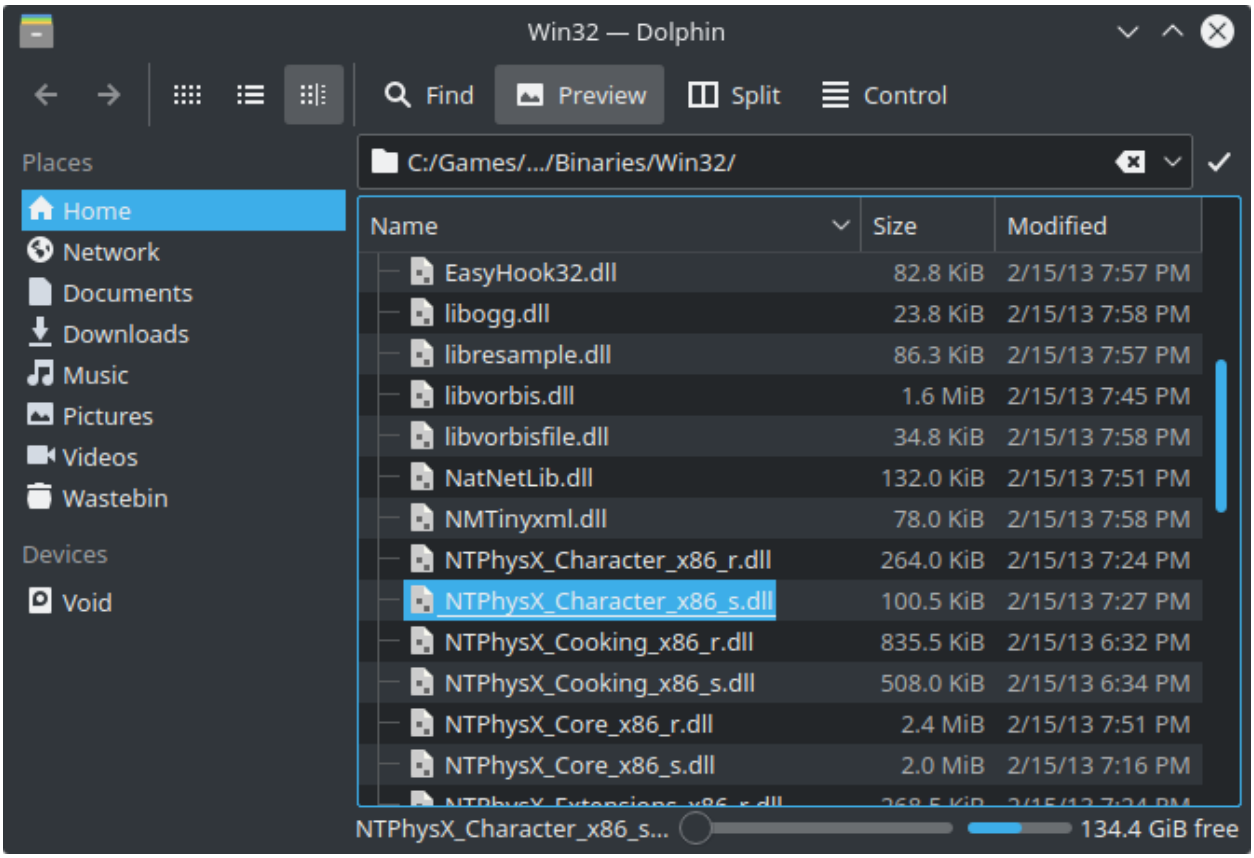
برنامه های با مدیریت حافظه خودکار (معمولا مجموعه زباله نامیده می شوند) نسبت به برنامه های غیرخودکار معمولا سرعت کمتری دارند. دلیل این مساله این است که این برنامه ها برای پاکسازی حافظه در بازه های خاص به زمان نیاز دارند.



تفاوت در کامپایل برنامه

معمولا برنامه های نوشته شده به زبان های C و C++ از بقیه سریعتر هستند. اکثرسیستم عامل ها با استفاده از این زبان ها و حتی کدهای اسمبلی سطح پایین تر نوشته می شوند. دلیل این مساله این است که این زبان ها بجای تفسیرشدن، کامپایل شده و سرعت بالاتری دارند.

در زبان های کامپایلی منبع برنامه پیش از اجرا به کدهای ماشینی ترجمه می شود. محصول این ترجمه کدهای باینری (فایل های dll) هستند که برای اجرای برنامه به فایل مربوطه لینک شده و فایل اجرایی را بوجود می آورند.

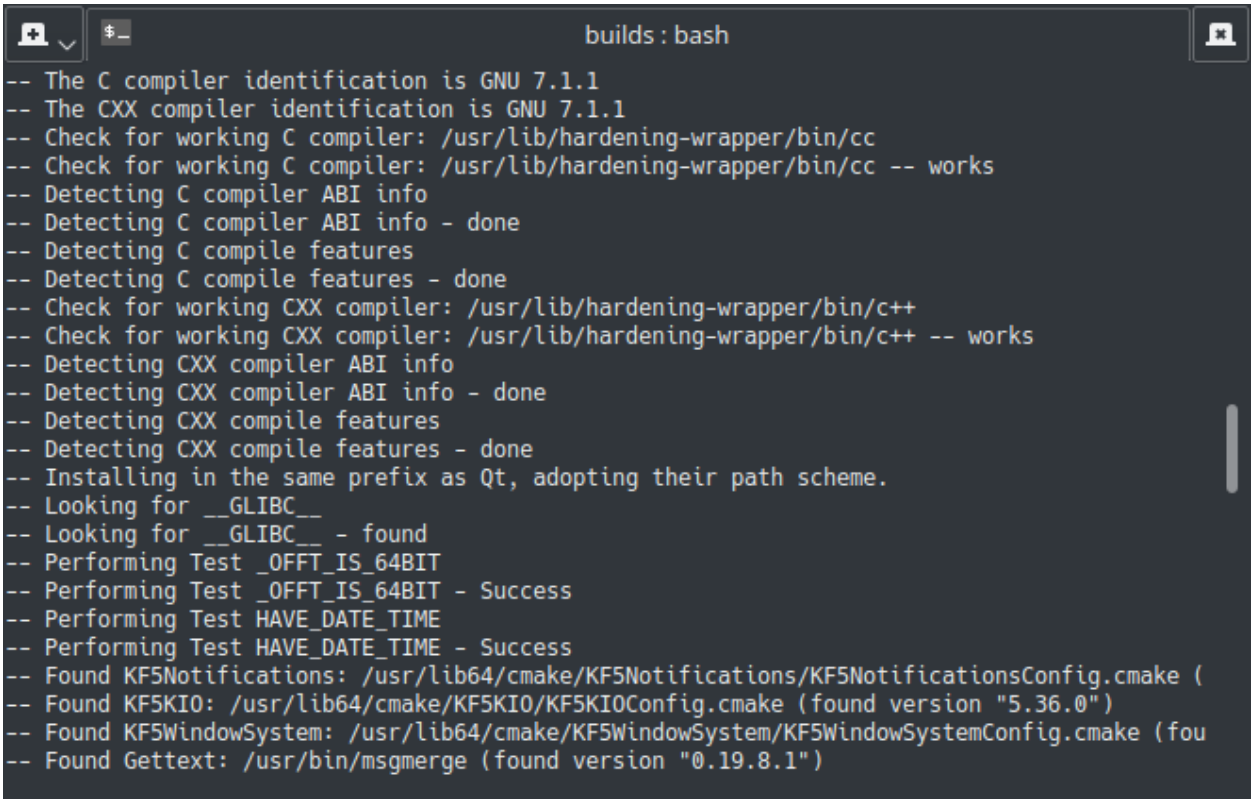


البته یکی از نواقص این روش این است که کامپایل برنامه های بزرگی مثل فایرفاکس ممکن است حدود سی دقیقه طول بکشد. خوشبختانه اکثر برنامه ها از قبل کامپایل شده و برای استفاده تنها لازم است انها را نصب کنیم.

از انجایی که برنامه نهایی به زبانی که برای کامپیوتر قابل درک است نوشته می شوند سرعت اجرای آن ها افزایش می یابد اما در زبان های تفسیری یک مرحله اضافی نیز انجام می شود. با این وجود کارایی تمام زبان های کامپایلی یکسان نیست.

کارایی کامپایلر

تمام کدها برای تبدیل شدن به زبانی که برای کامپیوتر قابل درک باشد (کد ماشین) باید از طریق برنامه ای که کامپایلر نامیده می شود ترجمه شوند. یک زبان می تواند چندین کامپایلر داشته باشد برای مثل از برنامه های GGC (مجموعه کامپایلر گنو) و Clang برای کامپیال کدهای نوشته شده به زبان C استفاده می شد.



نوع کامپایل برنامه ها می تواند بر سرعت اجرای آن ها تاثیر گذارباشد. برای مثال ممکن است دو کد یکسان که در کامپیلرهای با پیکربندی مختلف ترجمه شده اند، تفاوت اندکی در عملکرد داشته باشند.

این مساله نشان می دهد که زبان های کامپایل شده مختلف ممکن است سرعت اجرای متفاوتی داشته باشند. معمولا زبان های C و C++ نسبت به دیگر برنامه ها از سرعت بیشتری برخوردارند چون سال های زیادی از توسعه کامپالرهای این زبان گذشته و در این مدت بهبود زیادی پیدا کرده اند. زبان های دیگرهنوز به این حد تکامل پیدا نکرده اند.

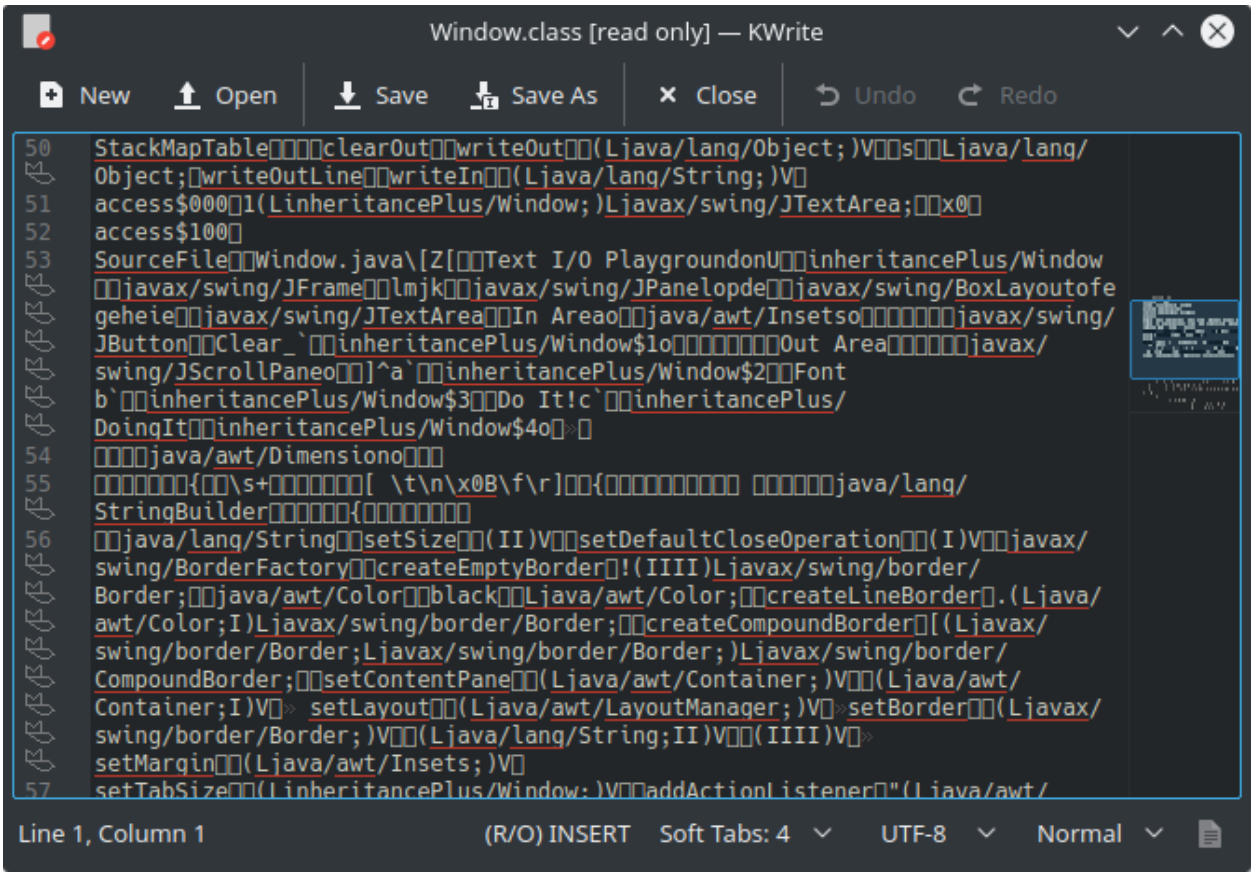
زبان های تفسیری و یک مرحله بیشتر

همانگونه که قبلا گفتیم زبان های تفسیری از قبل کامپایل نمی شوند بلکه هنگام اجرا از طریق برنامه جداگانه ای که مفسر نام دارد ترجمه می شوند. برای مثال برنامه های جاوا از طریق ماشین مجازی جاوا یا JVM اجرا می شوند.

از آنجایی که این برنامه ها به کامپایل اولیه نیاز ندارند، ساخت و آزمایش آن ها بسیار ساده تر است اما خود برنامه ها معمولا سرعت کمتری دارند. برنامه های مفسر، منبع برنامه را بصوت خط به خط اجرا می کنند که باعث کاهش سرعت می شود. علاوه براین خود مفسر هم برای اجرا شدن نیاز به زمان دارد.

ایجاد توازن با بایت کد

اکثر زبان های تفسیری برای افزایش سرعت از نوعی فرایند کامپایل استفاده می کنند. این برنامه پیش از اجرا به بایت کد ترجمه می شوند. بایت کد زبانی است که کارکردن با آن برای مفسرها ساده است. برای مثال زبان های جاوا و پایتون پیش از اجرا اینکار را بترتیب ازطریق ایجاد فایل های CLASS و PYC انجام می دهند.



در حقیقت این بایت کدها بین سرعت اجرا و سهولت توسعه یک برنامه توازن ایجاد میکنند. سرعت اجرای بایت کدها از زبان های تفسیری بیشتر بوده و توسعه آن ها از برنامه های کامپایلی ساده تر است. بایت کد در مقایسه با برنامه های کامپایل شده به کد ماشین از مزیت قابل پرتابل بودن نیز برخوردار است. اگر معماری CPU دارای مفسر باشد این برنامه ها در کامپیوتر اجرا می شوند.

در برخی زبان ها قابلیتی بنام کامپایل JIT (در لحظه) وجود دارد که بایت کدها را گرفته و آن ها را به کد ماشین تبدیل می کند. همانگونه که از نام این کامپایلر مشخص است این فرایند زمانی صورت می گیرد که برنامه اجرا می شود. هدف از اینکار افزایش سرعت اجرا به قیمت کاهش سرعت اولیه است چون در ابتدا باید بخش هایی از برنامه کامپایل شود.

زیاد سخت نگیرید

باوجود تمام مواردی که ذکر شد این تفاوت ها برای اکثر افراد اهمیت چندانی ندارند بویژه با در نظر گرفتن این نکته که سرعت کامپیوترها هر سال بیشتر و بیشتر می شود. هرچند بعضی از زبان های برنامه نویسی از بقیه سرعت بیشتری دارند اما مساله مهم این است که با زبان های کندتر هم می توان برنامه های خوبی نوشت.

اگر برنامه نویس باتجربه ای باشید می توانید روی افزایش کارایی برنامه تان کار کنید اما اگر تازه برنامه نویسی را شروع کرده اید بهتر است ابتدا روی یادگیری جنبه های مختلف تمرکز کنید. تا زمانی که برنامه ای نوشته نشده باشد مهم نیست یک زبان برنامه نویسی چقدر سریع یا کند است.

استفاده از ابزار درست برای نوشتن یک برنامه نیز اهیمت زیادی دارد. برای مثال برای توسعه یک سیستم عامل بهتر است از زبان اسمبلی استفاده شود چون سطح بسیار پایینی دارد اما استفاده از این زبان برای نرم افزارهای عادی توصیه نمی شود. زبان های دیگری نیز برای توسعه وب طراحی شده اند.