پایتون برای همه

نویسنده: دکتر چارلز سورنس
مترجم: دیانا مظهری

فصل 1

چرا باید نوشتن برنامه ها را یاد بگیرید؟

نوشتن برنامه (یا همان برنامه نویسی) یک کار بسیار خلاقانه است و موجب احساس موفقیت می شود. شما می توانید برای دلایل متفاوتی برنامه بنویسید، از کسب درآمد گرفته تا حل یک مسئله سخت بررسی داده ها تا سرگرمی و تفریح با کمک به دیگران برای حل مسئله. این کتاب فرض می کند که همه باید بتوانند برنامه بنویسند و وقتی که برنامه نویسی را یاد بگیرید، شما به کاری که با آن می خواهید انجام دهید پی میبرید.

اطراف ما در زندگی روزانه پر از کامپیوترهای گوناگون است، از لپ تاپ ها تا تلفن های همراه. ما می توانیم این کامپیوتر ها را به عنوان «دستیار شخصی» بدانیم که می تواند کاهایمان را برای ما انجام دهد. نرم افزار های امروزه به طوری ساخته شدهاند که به طور پیوسته از ما بپرسند، «چه کار دیگری می خواهی انجام بدهم؟»

برنامه نویس ها یک سیستم عامل و یک دسته اپلیکیشن به سخت‌افزار اضافه می کنند و ما یک دستیار شخصی دیجیتال به دست می آوریم که به طور قابل توجه مفید است و می تواند به ما در انجام کارهای زیادی کمک کند.

کامپیوترهای ما سریع هستند و دارای حافظهٔ عظیمی هستند و می توانستند برای ما خیلی سودمند باشند اگرما زبان آنها را می دانستیم تا با آنها حرف بزنیم و به آن بگوییم که میخواهیم چه کاری را برای ما انجام دهد. اگر ما این زبان را می دانستیم، می توانستیم به کامپیوتر بگوییم که چه کار های تکراری را از طرف ما انجام دهد. جالب است، فعالیت هایی که ماشین های کامپیوتری به بهترین حالت انجام می دهند، کار های مکرری هستند که انسان ها انجام آن را خسته کننده می دانند.

شکل 1.1: دستیار شخصی دیجیتال
شکل 1.2: حرف زدن برنامه نویس ها با شما

برای مثال، به سه پاراگراف اول این فصل نگاه کنید و به من پرتکرار ترین کلمه و اینکه چند بار استفاده شده را بگویید. با اینکه شما توانستید متن را در عرض چند ثانیه بخوانید و درک کنید، شمردن آنها کار بسیار خسته‌کننده‌ای است چون این کاری نیست که ذهن انسان برای انجام آن ساخته شده است. برای یک کامپیوتر، برعکس آن درست است، خواندن و درک متن از روی یک تکه کاغذ برای یک کامپیوتر سخت است ولی شمردن کلمات و اعلام این که چند بار این کلمه تکرار شده است برای کامپیوتر آسان است:

```
python words.py
Enter file: words.txt
to 16
```

«دستیار شخصی بررسی مشخصات» ما به سرعت به ما گفت که کلمه «و» 9 بار در سه پاراگراف اول این فصل استفاده شده است.

همان این نکته که کامپیوتر ها در انجام کار هایی که انسان ها توانایی انجام آن را ندارند، دلیل بر آن است که شما نیاز دارید که در صحبت کردن با «زبان کامپیوتر» مهارت پیدا کنید. هنگامی که شما این زبان جدید را یاد بگیرید، می توانید این فعالیت روزمره و تکراری را به عهده همکارتان بگذارید (همان کامپیوتر) و برای انجام کارهایی که به طور خاص مناسب شما هستند زمان بیشتری داشته باشید. شما به این همکاری خلاقیت، حس و توانایی کشف اضافه می کنید.

1.1 خلاقیت و انگیزه

با اینکه این کتاب برای برنامه نویس های ماهر نیست، برنامه نویسی جدی می تواند شغلی بسیار مفید هم از نظر مادی و هم از نظر شخصی باشد. ساختن برنامه های مفید، زیبا و هوشمند برای دیگران فعالیتی خلاقانه است. کامپیوتر شما یا همان دستیار شخصی دیجیتال معمولاً دارای چندین برنامه مختلف از چندین گروه برنامه نویسان به طوری که هر یک از آنها در حال رقابت برای توجه شما هستند. آنها بیشترین تلاش خود را می کنند تا نیاز های شما را برطرف کنند و تجربه کاربر مناسبی برای شما ایجاد کنند. در برخی شرایط، وقتی یک نرم‌افزاری را انتخاب می کنید، برنامه نویسان مستقیماً درآمد کسب می کنند.

اگر ما برنامه ها را نتیجه خلاقانه گروه هایی از برنامه نویس ها در نظر بگیریم، شکل مقابل ورژن منطقی تری از دستیار شخصی دیجیتال ما است:

اکنون، انگیزه اصلی ما برای درآمد یا رضایت کاربر ها نیست، ولی انگیزه ما این است که از وقتمان به خوبی برای کار با داده ها و مشخصاتی که در طول زندگی با آنها مواجهه می شویم. هنگامی که شروع می کنید، شما هم برنامه نویس و هم کاربر نهایی خواهید بود. با گذشت زمان با کسب مهارت بیشتر به عنوان برنامه نویس و برنامه نویسی باعث افزایش احساس خلاقیت شما می شود، شاید شما بیشتر به ساختن برنامه برای دیگران فکر کنید.

شکل 1.3: معماری سخت افزار

1.2 معماری سخت افزار کامپیوتر

قبل از این که ما یاد گرفتن زبانی که برای راهنمایی و حرف زدن با کامپیوتر برای ساخت نرم افزار را شروع کنیم، باید کمی درباره روش ساخت کامپیوتر ها یاد بگیریم. اگر شما تکه های کامپیوتر یا موبایلتان را جدا کنید، موارد زیر را داخل آن پیدا می کنید:

- *واحد پردازش مرکزی* (یا CPU) همان بخشی از کامپیوتر است که برای پرسیدن سوال «بعد چی؟» ساخته شده است. اگر سرعت کامپیوتر شما 3/0 گیگاهرتز باشد، یعنی پردازنده در هر ثانیه سه میلیارد بار می‌پرسد: «کار بعدی چیست؟». شما باید یاد بگیرید که سریع صحبت کنید تا بتوانید با پردازنده همگام شوید.
- *حافظهٔ اصلی* برای ذخیرهٔ اطلاعات و مشخصاتی است که CPU به سرعت به آن نیاز دارد. سرعت حافظهٔ اصلی تقریباً برابر CPU است. اما اطلاعات ذخیره شده در حافظهٔ اصلی پس از خاموش کردن کامپیوتر از بین می رود
- *حافظهٔ ثانویه* نیز برای ذخیرهٔ اطلاعات استفاده می شود ولی بسیار کندتر از حافظهٔ اصلی است. نقطه قوت حافظهٔ ثانویه این است که می تواند اطلاعات را حتی وقتی برقی به کامپیوتر نمی رسد. مثال‌هایی از حافظهٔ ثانویه شامل درایوهای دیسک یا حافظه فلش (که معمولاً در فلش‌مموری‌ها و پخش‌کننده‌های موسیقی قابل حمل یافت می‌شوند) است.

- *وسیله های ورودی و خروجی*، به زبان ساده صفحه های دیجیتال ما هستند، کیبورد، موس، میکروفون، بلندگو، پد لمسی و مانند آنها. اینها تمام روش هایی هستند که ما با کامپیوترها ارتباط برقرار می کنیم.
- امروزه همه کامپیوتر ها یک شبکه ارتباطی دارند تا اطلاعات را در طی یک شبکه بازیابی کنند. ما می توانیم از این شبکه به عنوان مکانی بسیار کند برای ذخیره و بازیابی اطلاعات داده هایی که شاید برای همیشه وجود نداشته باشد، استفاده کنیم. به عبارتی، این شبکه، شکل آهسته تر و بعضی مواقع غیر قابل اعتمادتر از حافظه ثانویه است.

با اینکه اکثر این نکات ریز که در این زمینه ها کار می کنند را بهتر است به عهده سازنده های کامپیوتر بگذاریم. داشتن واژه‌نامه‌ای در مورد بخش های مختلف کامپیوتر در عین نوشتن برنامه به ما کمک زیادی می کند.
شکل 1.4: تو کجا هستی؟

به عنوان یک برنامه نویس، وظیفه شما استفاده و تنظیم هر یک از این منابع است تا مسئله‌ای را که نیاز به حل دارید را با این داده های به دست آمده حل و بررسی کنید. به عنوان برنامه نویس شما به احتمال زیاد با CPU حرف می زنید و به آن می گویید که بعداً چه کاری را انجام دهد. گاهی شما به CPU می گویید که از حافظهٔ اصلی، حافظهٔ ثانویه، شبکه، یا وسایل ورودی و خروجی استفاده کنید.

باید همان کسی باشید که به سوال «کار بعدی چیست؟» پاسخ دهد. اما اگر شما را برای اینکه بتوانید در هر ثانیه سه میلیارد بار فرمانی را صادر کنید به اندازه 5 میلی متر برسانیم و داخل کامپیوتر قرار دهیم ، برای شما بسیار اذیت کننده می شود. پس در عوض، باید دستوراتتان را از قبل بنویسید. ما به این اطلاعاتی که از پیش نوشته می شوند، *برنامه* می گوییم و عمل نوشتن این راهنمایی ها و به خصوص درست نوشتن آنها *برنامه نویسی* نام دارد.

1.3 درک برنامه نویسی

در ادامهٔ این کتاب، تلاش می کنیم شما را به فردی تبدیل کنیم که در هنر برنامه نویسی مهارت یافته است. نهایتاً شما یک برنامه نویس خواهید شد - شاید نه یک برنامه نویس حرفه ای، اما حداقل توانایی کافی خواهید داشت که به یک مسئله در رابطه با داده یا اطلاعات نگاه کنید و برنامه ای بسازید که مسئله را حل کند.

به عبارتی، برای یک برنامه نویس شدن شما به دو مهارت نیاز دارید:

- اولاً، شما باید زبان برنامه نویسی را بدانید (پایتون) - باید با لغات و قواعد آن آشنایی کافی داشته باشید. شما باید نوشتن صحیح این کلمات را در این زبان جدید بدانید و روش نوشتن جملات با ساختاری مناسب در این زبان را یاد بگیرید.
- ثانیاً، باید یک داستان را تعریف کنید. در نوشتن داستان، شما واژه ها و جملات را در کنار هم قرار می دهید تا تصورات خود را به خواننده کتاب برسانید. ساختن این داستان نیازمند هنر و مهارت است و مهارت در داستان نویسی از راه نوشتن و دریافت بازخورد پیشرفت می کند. در برنامه نویسی، برنامه ی ما همان «داستان» و مسئله ای که شما سعی میکنید حل کنید همان «ایده» است.

هنگامی که شما یک زبان برنامه نویسی مانند پایتون را یاد بگیرید، یاد گرفتن زبانی دیگر مانند جاوا (Javascript) و سی پلاس پلاس (C++) برای شما آسان تر می شود. این زبان برنامه نویسی جدید دارای لغات و قواعد متفاوتی است ولی راه و روش حل مسئله در تمام زبان های برنامه نویسی یکسان خواهد بود.

شما «لغات» و «جملات» پایتون را به سرعت یاد خواهید گرفت. نوشتن برنامه‌ای منطقی و کامل برای حل یک مسئله، زمان بیشتری خواهد برد. ما برنامه نویسی را همانند نوشتن یاد می دهیم. ما با خواندن و توصیف برنامه شروع میکنیم، بعد برنامه های ساده می نویسیم، در نهایت به تدریج شروع به نوشتن برنامه های سخت‌تر و پیچیده‌تر می کنیم. در برهه‌ای از زمان شما الهام خود را می یابید و الگو ها را به تنهایی پیدا می کنید و بیشتر به طور طبیعی مشاهده می کنید که چگونه یک مسئله را درک کنید و برنامه ای برای حل آن بنویسید. وقتی به آن سطح می رسید، برنامه نویسی به فرایندی لذت‌بخش و خلاقانه تبدیل می شود.

ما با لغتنامه و ساختار پایتون شروع می کنیم. در یاد گرفتن صبور باشید که مثال های ساده یاد آور زمانی خواهد بود که برای اولین بار نوشتن را یاد می گیرید.

1.4 کلمات و جملات

برخلاف زبان های انسان، لغتنامه‌ی پایتون در واقع بسیار محدود است. به این «لغتنامه»، کلمه های خاص یا کلید واژه ها می گوییم. اینها کلماتی هستند که معانی بسیار خاصی برای پایتون دارند. وقتی پایتون این کلمات را در یک برنامه‌ی پایتون می بیند، اینها فقط و فقط یک معنی برای پایتون دارند. با گذر زمان، شما با نوشتن برنامه های مختلف، کلمات خاص خودتان را می سازید که معنی خاصی برای شما دارند که به آنها *متغیرها* می گویند. شما در انتخاب اسم های متغیر هایتان آزادی زیادی خواهید داشت، ولی نمیتوانید از هیچ یک از کلمات خاص پایتون به عنوان اسم متغیرهایتان استفاده کنید.

وقتی ما یک سگ را تربیت می کنیم، از کلمات خاصی مانند «بشین»، «بایست» و «بیار» استفاده می کنیم. هنگامی که با یک سگ صحبت می کنید و از هیچ یک از کلمات خاص استفاده نمی کنید، آنها فقط با یک نگاه پیچیده به شما نگاه می کنند تا اینکه از یک کلمه خاص استفاده کنید. به عنوان مثال، اگر شما بگویید:« ای کاش افراد بیشتری برای بهبود سلامت خود پیاده روی می کردند.»، چیزی که بیشتر سگ ها به احتمال زیاد می شنوند این است، « فلان فلان فلان پیاده‌روی فلان فلان فلان». این به همان دلیل است که «پیاده روی» یک کلمهٔ خاص در زبان سگ است. خیلی ها ممکن است بگویند که زبان بین انسان ها و گربه ها هیچ کلمات خاصی ندارد.

کلمات خاص (یا همان کلمات رزرو شده) در زبانی که انسان ها با پایتون صحبت می کنند شامل موارد زیر است:

False	await	else	import	pass
None	break	except	in	raise
True	class	finally	is	return
and	continue	for	lambda	try
as	def	from	nonlocal	while
assert	del	global	not	with
async	elif	if	or	yield

همین ها هستند، و برخلاف یک سگ، پایتون از ابتدا به طور کامل تربیت شده است. وقتی شما می گویید «try»، پایتون هر دفعه که آن را می گویید بدون اشتباه امتحان می کند.

ما درباره این کلمات رزرو شده و راه استفاده از آنها در زمان مناسب یاد می گیریم، اما اکنون بر روی معادل «صحبت کردن» (در زبان انسان به سگ) برای پایتون تمرکز می کنیم. نکته ی خوب در مورد فرمان دادن به پایتون که حرف بزند این است که ما حتی می توانیم با قرار دادن پیاممان داخل علامت های نقل قول ("") به آن بگوییم که چه چیزی پرینت کند:

```
print('Hello world!')
```

الان ما حتی اولین جملهٔ صحیح خود از نظر سینتاتیک را نوشتیم. جمله‌ی ما با تابع *print* شروع شده و به دنبال آن، رشته متنی(string) دلخواه قرار دارد که در علامت نقل قول تکی(single quotation mark) قرار دارد. رشته متن ها در عمل print داخل علامت های نقل قول قرار دارند. علامت های نقل قول تکی و دوتایی کار یکسانی می کنند؛ اکثر افراد از نقل‌قول تکی استفاده می‌کنند، مگر در مواردی که نقل‌قول تکی (که همان آپاستروف هم هست) درون رشته وجود داشته باشد.

1.5 صحبت کردن با پایتون

اکنون که ما یک کلمه و یک جمله از پایتون را می دانیم، باید بدانیم که چطور با پایتون مکالمه‌ای را آغاز کنیم تا مهارت های زبانی خود را امتحان کنیم.
قبل از اینکه بتوانید با پایتون صحبت کنید، شما باید ابتدا نرم‌افزار پایتون را دانلود کنید و یاد بگیرید که چگونه پایتون را به کار بیندازید. این کار مراحل زیادی برای این فصل دارد پس من پیشنهاد می کنم به سایت www.py4e.com مراجعه کنید که در آنجا من مراحل دانلود و نصب پایتون را از صفر تا صد با تصاویر برای سیستم های ویندوز و مک توضیح داده‌ام. در مقطعی، شما وارد یک ترمینال یا پنجره دستورات می شوید و واژهٔ python را تایپ می کنید و مترجم پایتون در حالت تعاملی (interactive mode) اجرا می شود و چیزی شبیه به زیر ظاهر خواهد شد:

```
Python 3.11.6 (main, Nov 2 2023, 04:39:43)
      [Clang 14.0.3 (clang-1403.0.22.14.1)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

این علامت <<< شیوهٔ مترجم پایتون برای پرسیدن این سوال از شماست: «می خواهی چه کاری در مرحله‌ی بعد انجام دهم؟» پایتون برای مکالمه با شما آماده است. تنها چیزی که لازم است بدانید این است که چگونه به زبان پایتون حرف بزنید.
بیایید تصور کنیم برای مثال شما حتی ساده‌ترین کلمات و جملات زبان پایتون را نمی دانید. شما شاید تمایل داشته باشید که این جمله‌ی استاندارد و معروفی که فضانوردان هنگام فرود روی سیاره‌ای دوردست و تلاش برای صحبت با ساکنان آن سیاره می گویند:

```
>>> I come in peace, please take me to your leader
File "<stdin>", line 1
      I come in peace take me tou your leader
        ^^^^
SyntaxError: invalid syntax
>>>
```

این خیلی خوب پیش نمی رود. مگر اینکه سریع دربارهٔ چیزی فکر کنید، ساکنان این سیاره به احتمال زیاد شما را با نیزه هایشان زخمی می کنند، روی سیخی می گذارند، روی آتش کباب می کنند و برای شام می خورند.
خوشبختانه شما یک نسخه از این کتاب را در سفرتان آورده‌اید و حالا به همین صفحه رجوع می کنی و دوباره تلاش می کنی:

```
>>> print('Hello world!')
Hello world!
```

الان این خیلی بهتر است، پس تلاش می کنید که بیشتر مکالمه کنید:

```
>>> print('You must be the legendary god that comes from the sky')
```

```
You must be the legendary god that comes from the sky
>>> print('We have been waiting for you for a long time')
We have been waiting for you for a long time
>>> print('Our legend says you will be very tasty with mustard')
Our legend says you will be very tasty with mustard
>>> print We will have a feast tonight unless you say
      File "<stdin>", line 1
            print 'We will have a feast tonight unless you say
                  ^
SyntaxError: unterminated string literal (detected at line 1)
>>>
```

این مکالمه خیلی خوب پیش می رفت تا اینکه شما کوچکترین اشتباه را در زبان پایتون کردید که پایتون نیزه ها را باری دیگر در آورد.

در این مرحله، باید فهمیده باشید که با اینکه پایتون به طور فوق العاده پیچیده، قدرتمند و نسبت به نحوهٔ نگارش حساس است، اما هوشمند نیست. در واقع شما فقط دارید با خودتان حرف می زنید ولی با نگارش درست.

به عبارتی، زمانی که از برنامه‌ای استفاده می کنید که توسط شخصی دیگر نوشته شده است، مکالمه بین شما و برنامه‌نویس های آن برنامه است و پایتون یک واسطه است. پایتون روشی است که سازندگان برنامه ها برای بیان اینکه مکالمه چگونه قرار است پیش برود استفاده می کنند. در چند فصل دیگر، شما هم یکی از آن برنامه‌نویس ها خواهید بود که از پایتون برای صحبت کردن با کاربر برنامهٔ خودتان به کار می برید.

قبل از اینکه اولین مکالمهٔ خود را با مترجم پایتون به پایان برسانیم، بهتر است راه مناسب «خداحافظ» گفتن را هنگام ارتباط با ساکنان سیارهٔ پایتون را بدانید:

```
>>> good-bye
Traceback (most recent call last):
      File "<stdin>", line 1, in <module>
NameError: name 'good' is not defined
>>> if you don't mind, I need to leave
      File "<stdin>", line 1
        if you don't mind, I need to leave
                  ^
SyntaxError: unterminated string literal (detected at line 1)
>>> quit()
```

متوجه خواهید شد که خطا در دو تلاش اول نادرست، با یکدیگر متفاوت است. خطای دوم متفاوت است چون if یک کلمه‌ی خاص است و پایتون این کلمه را دید و گمان کرد که ما سعی داشتیم چیزی بگوییم ولی نگارش جمله را اشتباه گرفتیم.

روش درست خداحافظی با پایتون این است که در علامت تعاملی <<< دستور quit() را وارد کنیم. احتمالاً زمان زیادی می برد تا آن را حدس بزنید، پس داشتن یک کتاب در دسترس کمک زیادی خواهد کرد.

1.6 اصلاحات: مترجم و کامپایلر

پایتون یک زبان سطح بالا است که طوری طراحی شده تا نسبتاً برای انسان ها خواندن و نوشتن آن ساده باشد و همچنین برای رایانه ها قابل خواندن و پردازش باشد. سایر زبان های سطح بالا شامل Java, C++, PHP, Ruby, Basic, JavaScript و غیره هستند. در واقع، سخت افزار داخل واحد پردازش مرکزی (CPU) هیچکدام از این زبان ها را نمیفهمد.
به زبانی که CPU می فهمد و درک می کند، زبان ماشین گفته می شود. زبان ماشینی بسیار ساده و در حقیقت نوشتن آن بسیار خسته کننده است، زیرا فقط شامل اعداد صفر و یک است:

```
00101000111010010010101010000001111
11100110000011101010010101101101
…
```

با توجه به اینکه فقط از صفر ها و یک ها ساخته شده است، زبان ماشینی بسیار ساده به نظر می رسد، ولی ساختار آن بسیار پیچیده تر و بسیار ضعیف تر از پایتون است. به همین سبب، برنامه‌نویس های کمتری با زبان ماشین می نویسند. در عوض، مترجم های مختلفی می سازیم که به ما اجازه می دهند که در این زبان های بالا سطح مانند پایتون یا جاوا اسکریپت بنویسیم و این مترجم ها برنامه ها را به زبان ماشینی تبدیل می کنند تا CPU بتواند آنها را اجرا کند.

از آنجا که زبان ماشینی به سخت‌افزار کامپیوتر مرتبط استفاده است، زبان ماشین بین سخت افزارهای مختلف قابل حمل نیست. برنامه هایی که با زبان های سطح بالا نوشته می شوند را می توان بین کامپیوترهای مختلف جابه‌جا کرد؛ این کار با استفاده از یک مترجم متفاوت روی دستگاه جدید یا با کامپایل مجدد کد برای ایجاد نسخه‌ای از برنامه به زبان ماشین مخصوص آن دستگاه انجام می شود.

این مترجم های زبان برنامه نویسی به دو گروه کلی تقسیم می شوند: (1) مترجم ها و (2) کامپایلرها.

یک مترجم، منبع برنامه را همانطور که برنامه نویس آن را نوشته، می خواند و آن را تجزیه می کند، سپس دستورات را بلافاصله تفسیر و اجرا می کند. پایتون یک زبان مترجم است و زمانی که به صورت تعاملی از پایتون استفاده می کنیم، می توانیم یک خط تایپ کنیم (یک جمله) و پایتون بلافاصله آن را پردازش می کند و آماده است تا خط بعدی را از ما دریافت کند.

بعضی از خط های پایتون به پایتون می گویند که شما از آن می خواهید که بعضی مقادیر را برای بعد به یاد داشته باشد. ما می توانیم برای این مقدار یک نام انتخاب کنیم تا بتوانیم بعداً با استفاده از آن نام نمادین به مقدار ذخیره شده دست یابیم. ما از اصطلاح متغیر (variable) برای اشاره به این برچسب ها استفاده می کنیم که به داده های ذخیره شده ارجاع می دهند.

```
>>> x = 6
>>> print(x)
6
>>> y = x * 7
>>> print(y)
42
>>>
```

در این مثال، ما از پایتون می خواهیم که مقدار شش را به یاد داشته باشد و از برچسب x برای آن استفاده کند که بعد بتوانیم این مقدار را بازیابیم و ما می توانیم با استفاده از print این را تایید کنیم که پایتون این مقدار را به یاد دارد و

ذخیره کرده است. بعد به پایتون دستور می دهیم که x را بازیابد و در هفت ضرب کند و مقدار جدید به دست آمده را داخل y قرار دهد . بعد از آن می خواهیم که این مقدار داخل y را پرینت کند.

با اینکه این فرمان ها را خط به‌خط در پایتون وارد می کنیم، پایتون آنها را به عنوان یک دنبالهٔ مرتب از دستورات در نظر می گیرد، به طوری که دستورات بعدی می توانند به اطلاعاتی که در دستورات قبلی ایجاد شده‌اند دسترسی پیدا کنند. در واقع، ما در حال نوشتن اولین پاراگراف سادهٔ خود با چهار جملهٔ معنادار و منطقی هستیم.

همانطور که قبلاً نشان داده شد، توانایی یک مکالمه متعاملی در ذات یک مترجم قرار دارد. یک کامپایلر برای این که فرآیندی را اجرا کند تا کد منبع سطح بالا را به زبان ماشین تبدیل کند، باید

برای اینکه یک کامپایلر فرآیندی را اجرا کند تا کد منبع سطح بالا را به زبان ماشین تبدیل کند، باید کل برنامه را به صورت یک فایل کامل دریافت کند. سپس این کامپایلر، زبان ماشین نهایی را داخل فایل می گذارد تا بعداً اجرا شود.

اگر از سیستم‌عامل ویندوز استفاده می کنید، معمولاً برنامه هایی که به زبان ماشین قابل اجرا هستند پسوندی مانند ".exe" یا ".dll" دارند که به ترتیب معنای قابل اجرا (executable) و کتابخانه‌ی پیوند پویا (dynamic link library) هستند. در لینوکس و مکینتاش، پسوند خاصی برای مشخص کردن فایل های قابل اجرا وجود ندارد.

اگر شما یک فایل قابل اجرا را در یک ویرایشگر متن باز کنید، با چنین چیزی غیرقابل درک و خواندن مواجه می شوید:

```
^?ELF^A^A^A^@^@^@^@^@^@^@^@^@^B^@^C^@^A^@^@^@\xa0\x82
^D^H4^@^@^@\x90^]^@^@^@^@^@^@4^@ ^@^G^@(^@$^@!^@^F^@
^@^@4^@^@^@4\x80^D^H4\x80^D^H\xe0^@^@^@\xe0^@^@^@^E
^@^@^@^D^@^@^@^C^@^@^@^T^A^@^@^T\x81^D^H^T\x81^D^H^S
^@^@^@^S^@^@^@^D^@^@^@^A^@^@^@^A\^D^HQVhT\x83^D^H\xe8
....
```

خواندن و نوشتن زبان ماشینی اصلا کار ساده‌ای نیست، پس خوب است که مترجم ها و کامپایلرهایی را داریم که به ما اجازه می دهند در زبان های سطح بالا مانند پایتون یا سی بنویسیم.

هم‌اکنون در این نقطه از بحث ما دربارهٔ کامپایلرها و مترجم ها، به احتمال زیاد شما دربارهٔ مترجم خود پایتون فکر می کنید. در چه زبانی نوشته شده است؟ آیا در زبانی فشرده نوشته شده است؟ وقتی ما «python» تایپ می کنیم، دقیقا چه اتفاقی می افتد؟
مترجم پایتون در یک زبان سطح بالایی به نام C نوشته شده است. شما می توانید به کد منبع واقعی پایتون با مراجعه کردن به www.python.com نگاه کنید و آن را بررسی کنید. پس پایتون خود یک برنامه است و به کد ماشین تبدیل و فشرده شده است. وقتی شما پایتون را داخل کامپیوتر خود نصب کردید، یک ورژن ترجمه شده از برنامهٔ پایتون به زبان ماشین روی سیستم خود کپی کرده‌اید. در ویندوز، کد ماشین قابل اجرا برای پایتون به احتمال زیاد داخل فایلی مشابه به این است:

```
C:\Python35\python.exe
```

1.7 نوشتن یک برنامه

تایپ کردن فرمان های مختلف به مترجم پایتون راهی عالی برای امتحان و آزمایش امکانات پایتون است ولی این کار برای مسئله های پیچیده تر توصیه نمیشود.
وقتی میخواهیم یک برنامه بنویسیم از یک ویرایشگر متن استفاده می کنیم تا فرمان های پایتون را در فایلی بنویسیم که به آن *اسکریپت* گفته می شود. در نتیجه، اسکریپت های پایتون با اسم py. تمام می شوند.

برای اجرای اسکریپت، شما باید به مترجم پایتون اسم فایل را بگویید. در یک پنجرهٔ فرمان (command window)

شما اسم python hello.py را به شکل زیر تایپ کنید:

```
$ cat hello.py
 print('Hello world!')
 $ python hello.py
 Hello world!
```

علامت «$» نشاندهندهٔ اعلان سیستم‌عامل است و دستور «cat hello.py» به ما نشان می دهد که فایل «hello.py» شامل برنامهٔ تک خطی پایتون برای یک رشته است. مترجم پایتون را فعال می کنیم و دستور می دهیم که کد منبع فایل «hello.py» را بخواند به جای اینکه منتظر بماند تا خطوط کد پایتون را یکی‌یکی از ما دریافت کند.

متوجه می شوید که نیازی به استفاده از ()quit در انتهای فایل برنامهٔ پایتون نبود. زمانی که پایتون کد منبع شما را از یک فایل می خواند، می داند زمانی که به انتهای فایل می رسد باید توقف کند.

1.8 برنامه چیست؟

تعریف یک برنامه در ساده‌ترین حالت، دنباله‌ای از دستورات پایتون است که با دقت طراحی شده‌اند تا کاری را انجام دهند. حتی اسکریپت سادهٔ hello.py یک برنامه است. یک برنامهٔ تک خطی است اگرچه چنان کاربردی نیست، اما در دقیق‌ترین تعریف، این یک برنامه‌ی پایتون است.

به احتمال زیاد راحترین راه فهمیدن این که برنامه چیست، این است که اول به مسئله‌ای فکر کنیم که یک برنامه می تواند برای حل آن ساخته شود و بعد به برنامه‌ای نگاه کنیم که آن مسئله را حل می کند.

فرض کنیم شما دربارهٔ محاسبات اجتماعی از روی پست های فیسبوک(Facebook) تحقیق می کنید و می خواهید پرتکرارترین کلمه در سری مختلف پست ها را بدانید. برای این کار می توانید همهٔ آن پست ها را پرینت کنید و متن همهٔ آنها را یکی‌یکی بررسی کنید تا پرتکرارترین کلمه را پیدا کنید، اما این کار زمان بسیار زیادی می کشد و احتمال خطای بسیار بالایی دارد. کار عاقلانه این می باشد که یک برنامهٔ پایتون بنویسید تا بتوانید این کار را به سرعت زیاد و دقیق انجام دهید و آخر هفتهٔ خود را مشغول به تفریح باشید.

برای مثال، به متن زیر دربارهٔ یک دلقک و یک ماشین دقت کنید. به این متن نگاه کنید و پرتکرارترین کلمه را بیابید و پیدا کنید این کلمه چند بار تکرار شده است.

```
the clown ran after the car and the car ran into the tent and the tent fell
down on the clown and the car
```

سپس تصور کنید که شما در حال انجام این کار با بررسی میلیون ها نوشته هستید. در حقیقت، یاد گرفتن پایتون و نوشتن برنامهٔ پایتون برای شمردن این کلمات سریع تر از انجام این کار به صورت دستی است.

خبر بهتر این است که من هم‌اکنون برنامه‌ای نوشته‌ام تا پرتکرارترین کلمه را در فایل نوشته بیابد. آن را نوشتم، امتحان کردم و الان هم در اختیار شما قرار می دهم تا از آن استفاده کنید و از زمان خود صرفه‌جویی کنید:

```
name = input('Enter file: ')
handle = open(name, 'r')
counts = dict()

for line in handle:
      words = line.split()
      for word in words:
            counts[word] = counts.get(word, 0) + 1

bigcount = None
bigword = None
for word, count in list(counts.items()):
      if bigcount is None or count > bigcount:
      bigword = word
      bigcount = count

print(bigword, bigcount)

# Code: https://www.py4e.com/code3/words.py
```

حتی لازم نیست به پایتون تسلط داشته باشید تا از این برنامه استفاده کنید. باید فصل 10 این کتاب را بخوانید که تمام تکنیک هایی که در نوشتن این برنامه به کار رفته را به طور کامل بفهمید. شما کاربر نهایی این برنامه هستید، فقط از برنامه استفاده می کنید و در برابر قدرت و هوش آن شگفت‌زده می شوید که چگونه شما را از انجام کاری خسته کننده نجات داد. شما فقط این کد را داخل فایلی به نام words.py و آن را اجرا می کنید یا کد منبع (source code) آن را از http://www.py4e.com/code3/ دانلود کنید و آن را اجرا کنید.

این یک مثال بسیار خوبی برای درک چگونگی پایتون و زبان پایتون که به صورت رابط میان شما (کاربر) و من (برنامه نویس) عمل می کند. پایتون راهی مناسب برای تبادل دنباله هایی از دستورات مفید (یعنی برنامه ها) در یک زبان مشترک و قابل استفاده برای هر کسی که برنامهٔ پایتون را در کامپیوتر خود نصب کند، است. پس هیچ یک از ما در حال صحبت با پایتون نیستیم، در واقع ما از طریق پایتون با هم حرف می زنیم.

1.9 اجزای سازنده برنامه ها

در چند فصل آینده، بیشتر دربارهٔ واژه‌نامه، ساختار جمله، ساختار پاراگراف و ساختار داستان پایتون یاد میگیریم. دربارهٔ توانایی های جالب پایتون یاد میگیریم و چگونه این توانایی ها را در کنار هم قرار دهیم تا برنامه های مفید بسازیم.
الگو های مفهومی ساده‌ای وجود دارد که ما از آن برای ساختن و طراحی برنامه ها استفاده می کنیم. این ساختار ها فقط برای برنامه های پایتون نیستند، اینها همه بخشی از هر زبان برنامه نویسی هستند؛ از زبان ماشین گرفته تا زبان های سطح بالا.

ورودی از «دنیای خارجی» داده جمع‌آوری کنید. این کار می تواند بررسی داده ها و اطلاعات از یک فایل باشد یا حتی نوعی سنسور مانند میکروفون یا جی پی اس. در برنامه های اولیه ما، ورودی ما از طریق دادهٔ تایپ شده از طرف کاربر با کیبورد به دست می آید.

خروجی نتیجه های برنامه را روی صفحه‌ای نشان دهید یا آنها را در یک فایل ذخیره کنید یا شاید هم آنها را به یک دستگاه بنویسید مانند بلندگو تا موسیقی پخش کند یا متنی را بخواند.

اجرای ترتیبی دستورات را یکی پس از دیگری به صورتی که در اسکریپت ترتیب داده شده‌اند اجرا کنید.

اجرای شرطی شرایط خاص را بررسی کنید و آن را اجرا کنید و یا دنباله یا دستوری را رد کنید.

اجرای تکراری بعضی از دستورات را به صورت تکراری اجرا کنید، معمولاً با مقداری تنوع و تغییر.
استفاده مجدد دسته‌ای از فرمان ها را یک بار بنویسید و به آنها اسمی دهید، سپس در صورت نیاز، آن دستورات را در بخش های مختلف برنامه‌ی خود دوباره استفاده کنید.

این موضوع تقریباً بیش از حد آسان به نظر می رسد تا واقعیت داشته باشد، و البته هیچ وقت آنقدر ساده و آسان نیست. همانند آن است که بگوییم راه رفتن فقط «گذاشتن یک پا جلوی دیگری است». «هنر» نوشتن برنامه چندین بار قرار دادن و ارتباط دادن این عناصر اصلی در کنار هم است تا چیزی مفید برای کاربران آن بسازد.

برنامهٔ شمردن کلمات بالا به طور مستقیم از همهٔ این الگو ها استفاده میکند به جز یکی.

1.10 چه مشکلی ممکن است پیش بیاید؟

همانطور که مکالمات اول خود را با پایتون دیدیم، ما باید هنگام مکالمه با پایتون بسیار دقیق کد پایتون را بنویسیم. کوچکترین انحراف یا اشتباه باعث می شود پایتون بررسی ادامه‌ی برنامه‌ی شما را متوقف کند.

برنامه نویسان مبتدی از این نکته که پایتون هیچ جایی برای اشتباه نمیدهد، این را برداشت می کنند که پایتون سختگیر، بی رحم و خشن است. درحالی که پایتون از دیگران خوشش می آید و با آنها خوب است، فکر می کنند پایتون آنها را شخصاً می شناسد و از آنها کینه به دل دارد. به خاطر این کینه، پایتون برنامه هایی که به نظر ما کاملاً صحیح نوشته شده‌اند، به عنوان «نامناسب» رد می کند فقط برای اینکه می خواهد ما را اذیت کند.

```
>>> primt ‘Hello world!’
File "<stdin>", line 1
      primt ‘Hello world!’
            ^^^^^^^^^^^^^^
SyntaxError: invalid syntax

>>> primt (‘Hello world’)
Traceback (most recent call last):
     File "<stdin>", line 1, in

NameError: name ‘primt’ is not defined. Did you mean: ‘print’?

>>> I hate you Python!
File "<stdin>", line 1
      I hate you Python!
        ^^^^
SyntaxError: invalid syntax
>>> if you come out of there, I would teach you a lesson
File "<stdin>", line 1
      if you come out of there, I would teach you a lesson
             ^^^^
SyntaxError: invalid syntax
>>>
```

چیزی با دعوا کردن با پایتون به دست نمی آید. پایتون فقط یه وسیله است. هیچ احساساتی ندارد و خوشحال و حاضر این است که به فرمان و دستورات شما عمل کند. پیام های ارور آن سختگیر و بی رحم به نظر می رسند ولی آنها فقط

فریادهای پایتون برای کمک هستند. به چیزهایی که شما نوشته‌اید نگاه کرده است و درک نمی کند شما چه چیزی نوشته‌اید.

پایتون بیشتر شبیه یک سگ است، شما را بی نهایت دوست دارد، چند کلیدواژه‌ای دارد که شما را درک کند، با چشم‌های بامزه به شما نگاه می کند (<<<) و منتظر شما است که چیزی که توان درک آن را دارد را بگویید. وقتی پایتون می گوید «`SyntaxError: invalid syntax`»، دارد دمش را تکان می دهد و می گوید:«ظاهراً تو یه چیزی به من میگی ولی من آن را نمیفهمم ولی لطفا با من حرف بزن (<<<).»

با گذر زمان که برنامه های شما پیشرفته و پیچیده‌تر می شود، شما با سه نوع خطا (Error) مواجهه می شوید:

خطا های سینکاکس (Syntax errors) اینها اولین خطاهایی هستند که با آنها مواجهه می شوید و همچنین راحت‌ترین آنها برای رفع هستند. یک خطای سینتاکس به این معنی است که شما قوانین «قواعد» پایتون را زیر پا گذاشته‌اید. پایتون تلاش زیادی می کند که خط و کاراکتری که خطای شما را تشخیص داده را به طور دقیق نشان دهد. تنها نکته‌ی قابل توجه دربارهٔ خطا های سینتاکس این است که گاهی خطایی که نیاز به تصحیح است قبل تر از جایی است که پایتون متوجه اشتباه شده و خطا نشان می دهد. پس خط و کاراکتری که پایتون نشان می دهد میتواند نقطهٔ شروع بررسی شما باشد.

خطا های منطقی (Logic errors) یک خطای منطقی زمانی است که برنامهٔ شما سینتاکس خوبی دارد ولی خطایی در ترتیب دستورات یا شاید خطایی در چگونگی ارتباط دستورات شما به یکدیگر وجود دارد. نمونه‌ای مناسب برای یک خطای منطقی می تواند این باشد که «از بطری خود آب بخور، آن را داخل کیف خود قرار بده، به طرف کتابخانه برو، بعد درِ بطری را ببند.»

خطا های معنایی (Sematic errors) خطای معنایی هنگامی است که توصیف شما از مراحل انجام کار از نظر سینتاکس کاملاً درست و به ترتیب صحیح است اما اشتباهی در خود منطق برنامه وجود دارد. برنامه به طور کامل درست است ولی کاری که شما میخواهید را انجام نمی دهد. مثالی مناسب می تواند این باشد؛ اگر شما به فردی آدرس یک رستوران را می دهید و می گویید:«...وقتی به تقاطع کنار پمپ گاز رسیدی، به چپ بپیچ و یک مایل برو و رستوران یک ساختمان قرمز در سمت چپ است.» ولی دوست شما دیر کرده و به شما زنگ می زند و می گوید که در مزرعه‌ای هستند و پشت یک طویله راه می روند و هیچ رستورانی نمیبینند. بعد شما می گویید:« در پمپ گاز به چپ پیچیدی یا راست؟» و بعد می گویند:«من آدرسی که تو دادی را به طور دقیق دنبال کردم، آنها را نوشتم، نوشته در پمپ گاز به چپ بپیچ و یک مایل جلو برو.» سپس شما می گویید:«من خیلی متأسفم، چون با اینکه راهنمایی های من از نظر سینتاتیک کاملا درست است ولی متأسفانه خطایی معنایی کوچک دارند که تشخیص داده نشد.»

در هر سه نوع از این خطا ها، پایتون بیشترین تلاش خود را می کند تا دستوراتی که شما داده‌اید را به طور دقیق اجرا کند.

1.11 دیباگ کردن (خطاگیری)

وقتی پایتون یک ارور (خطا) می دهد یا حتی زمانی که جوابی متفاوت از آنچه شما به منظور داشتید بدهد، از آن به بعد شکار همان خطا شروع می شود. دیباگ کردن (رفع خطاها) همان روند یافت منبع خطا در کد است. هنگامی که شما در حال دیباگ برنامه‌ای هستید، به خصوص خطایی سخت، چهار چیزی است که باید به آنها توجه کنید:

خواندن کد خودتان آزمایش کنید، دوباره به خودتان بخوانید، سپس چک کنید که همان چیزی را می گوید که شما منظورش را داشتید.

اجرا کردن با ایجاد تغییرات مختلف و اجرای ورژن های متمایز آن را امتحان کنید. اغلب اگر چیزی را در جای مناسب برنامه نمایش دهید، مشکل به طور کامل واضح می شود، اما بعضی اوقات باید زمانی را صرف کنید تا ساختار اولیه را بسازید.

تفکر عمیق کمی فکر کنید! چه نوع خطایی است: سینتاکس، ران تایم (زمان اجرا)، یا معنایی؟ چه اطلاعاتی را میتوانید از پیام های خطا یا خروجی برنامه دریافت کنید؟ چه نوع خطایی می تواند مشکلی که شما با آن مواجهه می شوید را ایجاد کند؟ قبل از اینکه این مشکل ظاهر شود، آخرین چیزی که عوض کرده بودید چه بود؟

عقب‌نشینی کردن مواقعی می رسد که بهترین کار این است که کنار بکشید، تغییراتی که اخیراً انجام داده‌اید را پاک کنید تا زمانی که به برنامه‌ای برسید که کار می کند و شما درک می کنید. سپس می توانید دوباره آن را بسازید.

برنامه نویسان تازه کار گاهی در این فعالیت ها درگیر می شوند و از بقیه غافل می شوند. پیدا کردن یک باگ (خطا) دشوار به خواندن، اجرا کردن، تفکر عمیق و گاهی به کنار کشیدن و عقب نشینی نیاز دارد. اگر در یکی از این فعالیت ها مواجهه مشکل شوید، بقیه را امتحان کنید. هر فعالیتی نوع خاص خطای خودش را به همراه دارد.

برای مثال، خواندن کد شاید زمانی به کار بیاید که مشکل برنامه خطای تایپی است، ولی اگر مشکل مفهمومی است این روش بی فایده است. اگر شما نفهمید که خطای شما باعث چه می شود، حتی اگر 100 دفعه هم آن را بخوانید، خطا را پیدا نمی کنید چون مشکل و خطا در دانسته های شما است.

آزمایش کردن می تواند کمک کند، به خصوص اگر شما تست های ساده و کوچکی اجرا کنید. اما اگر آزمایشهایتان را بدون فکر کردن و خواندن کدتان اجرا کنید، شاید دچار الگویی شوید که من به آن «برنامه‌نویسی با گام های تصادفی» می گویم، که همان روند ایجاد تغییرات تصادفی تا زمانی که برنامه کار درست را انجام دهد. طبیعاً میتوان فهمید که برنامه‌نویسی تصادفی زمان زیادی طول می کشد.

باید برای فکر کردن وقت بگذارید. دیباگ کردن (خطاگیری) شبیه یک علم تجربی است. شما حداقل به یک فرضیه دربارهٔ اینکه مشکل چیست نیاز دارید. اگر دو یا چند احتمال وجود دارد، سعی کنید دربارهٔ آزمایشی فکر کنید که یکی از آنها را حذف کند.

استراحت کردن به فکر کردن کمک می کند. صحبت کردن هم همینطور. اگر مشکل را به دیگران توضیح دهید (یا حتی خودتان)، گاهی مشکل را حتی قبل از تمام کردن سوالتان پیدا می کنید.

ولی اگر خطاهای زیادی وجود داشته باشد، حتی بهترین تکنیک های دیباگ هم دچار مشکل می شود یا اگر کدی که می نویسید خیلی بزرگ و پیچیده باشد. بعضی اوقات بهترین گزینه همان کنار کشیدن است، برنامه را تا زمانی که به درستی کار کند و شما آن را درک کنید، ساده کنید.

برنامه‌نویس های تازه‌کار معمولاً از عقبنشینی خودداری می کنند، چون تحمل نوشتن دوبارهٔ برنامه را ندارد (حتی اگر غلط باشد). حتی می توانید برنامه را قبل از اینکه پاک کنید، آن را به فایلی دیگر کپی کنید (اگر این کار شما را راضی می کند). می توانید تکه های آن را یکی‌یکی اضافه کنید.

1.12 روند یاد گیری

درصورتی که درادامهٔ کتاب پیش می روید، اگر موضوعات در نگاه اول با هم بی ربط به نظر برسند نترسید. وقتی شما اول حرف زدن را یاد می گرفتید، ایرادی نداشت که شما در چند سال اول حرف زدن صدا های بامزه‌ای میدادید و اگر شش ماه طول کشید که کلمات ساده به جملات ساده تبدیل شود و 5 یا 6 سال طول کشید که این جملات به پاراگراف ها تبدیل شد و چند سال دیگر تا بتوانید به تنهایی یک داستان کوتاه بنویسید هم باز عیب نبود.
ما میخواهیم شما پایتون را با سرعتی بیشتر یاد بگیرید، به همین سبب همه را در کنار هم در طول چند فصل آینده یاد می دهیم. اما این روند همانند یاد گرفتن یک زبان جدید زمان می برد که آن را هضم و درک کنید و بعد به طور طبیعی آن را درک کنید. این موضوع باعث کمی تعجب می شود، چرا که ما مرتب به مباحث سر می زنیم و دوباره به آنها برمی‌گردیم تا همزمان که در حال تعریف بخش های کوچک هستیم، شما بتوانید تصویر کلی را هم ببینید. درحالی که این کتاب به صورت خطی نوشته شده است و اگر در دوره‌ای شرکت می کنید آن هم به صورت خطی پیش می رود، شما می توانید با مطالب این کتاب رویکردی غیرخطی هم داشته باشید. به مطالب پس و پیش نگاه کنید و آن را جزئی بخوانید. با روخوانی جزئی مطالب پیشرفته بدون درک کامل جزئیات، می توانید دلیل برنامه‌نویسی را بهتر بفهمید. با مرور مطالب قبلی و حتی حل دوبارهٔ مسائل قبلی؛ پی می برید که بسیاری از مطالبی که قبلاً خواندید را می فهمید و آنها را درک می کنید حتی اگر مطالبی که درحال‌حاضر به آنها خیره شده‌اید غیرقابل درک به نظر برسد.
معمولاً وقتی اولین زبان برنامه‌نویسی را یاد میگیرید، لحظات خاص «آهان!» وجود دارد؛ لحظاتی که سرتان را از روی سنگی که با چکش و قلم می کوبیدید بلند می کنید و یک قدم عقب‌تر می‌روید و متوجه می شوید که واقعاً در حال ساختن یک مجسمه‌ای زیبا هستید.
اگر چیزی به‌ویژه سخت به نظر می رسد، معمولاً فایده‌ای ندارد که تمام شب را بیدار بمانید و به آن نگاه کنید. استراحت کنید، چرتی بزنید، چیزی بخورید، به کسی توضیح دهید که با چه چیزی مشکل دارید (یا شاید به سگتان)، بعد با ذهنی تازه به آن برگردید. به شما اطمینان می دهم که وقتی مفاهیم برنامه‌نویسی این کتاب را یاد بگیرید، به گذشته نگاه خواهید کرد و خواهید دید که همه‌چیز واقعاً ساده و زیبا بوده و فقط کمی فرصت لازم داشتید تا آن‌ها را درک و جذب کنید.

1.13 واژه‌نامه

باگ خطایی در یک برنامه

واحد پردازش مرکزی قلب هر کامپیوتر. این همان چیزی است که نرم‌افزاری که ما مینویسیم را اجرا می کند؛ همچنین به آن «CPU» یا «processor» هم می گویند.

کامپایل (compile) به عمل ترجمهٔ همزمان زبانی سطح بالا به زبانی سطح پایین درجهت آمادگی برای اجرای بعدی.

زبان سطح بالا (High-level language) یک زبان برنامه‌نویسی مانند پایتون که برای ساده شدن خواندن و نوشتن طراحی شده.

حالت تعاملی (interactive mode) روشی برای استفاده از مترجم پایتون با تایپ فرمان ها و عبارات بعد از دیدن پرامپت (نشانهٔ ورودی).

تفسیر و اجرا کردن(interpret) اجرای برنامه‌ای در زبانی سطح بالا با ترجمهٔ آن به طور خطی پس از دیگری.

زبان سطح پایین(low-level language) یک زبان برنامه‌نویسی که به خاطر ساده کردن اجرا برای کامپیوتر طراحی شده؛ همچنین به آن «کد ماشین» یا «زبان اسمبلی».

کد ماشین (machine code) پایین‌ترین سطح زبان برای نرم‌افزار، همان زبانی است که به‌طور مستقیم از طرف واحد پردازش مرکزی (CPU) اجرا می شود.

حافظهٔ اصلی(main memory) برنامه ها و داده ها را ذخیره می کند. حافظهٔ اصلی تمام اطلاعاتش را پس از خاموش شدن از دست می دهد.

تجزیه کردن (parse) آزمایش و تحلیل برنامه و بررسی ساختار سینتاتیک آن.
احتمال ویژگی یک برنامه که میتواند برای انواع مختلف کامپیوتر اجرا شود.
تابع پرینت یک دستوری که باعث میشود اجراگر پایتون ارزشی را روی صفحه ظاهر کند.
حل مسئله روند طراحی یک مسئله، پیدا کردن راه حل و بیان آن راه حل.
برنامه(program) دسته‌ای از دستورات که یک فرآیند محاسباتی را تعریف میکند.
پرامپت یا نشانهٔ ورودی(prompt) زمانی که یک برنامه پیامی را نشان می دهد و توقف می کند که کاربر به آن ورودی تایپ کند.
حافظهٔ ثانویه(secondary memory) برنامه‌ها و داده ها را ذخیره می کند و اطلاعات آن را حفظ می کند حتی زمانی که خاموش است. به طور کلی کندتر از حافظهٔ اصلی است. مثال هایی از حافظهٔ ثانویه شامل درایوهای دیسک و حافظه فلش در فلش‌مموری‌های USB.
سماتیک‌ها یا همان معناشناسی(sematic) معنی یک برنامه
خطای معنایی(sematic error) خطایی در یک برنامه که باعث می شود کاری را خارج از آنچه برنامه‌نویس منظور داشت انجام دهد.
کد منبع(source code) برنامه‌ای در زبان سطح بالا.

1.14 تمرین ها

تمرین 1: کاربرد حافظهٔ ثانویه در یک کامپیوتر چیست؟
الف) تمام محاسبات و منطق برنامه را اجرا کند.
ب) حفظ کردن صفحات وب در اینترنت.
ج) ذخیرهٔ بلندمرتبه اطلاعات، حتی فراتر از خاموش و روشن شدن دستگاه.
د) دریافت ورودی از کاربر
تمرین 2: برنامه چیست؟
تمرین 3:تفاوت بین یک کامپایلر و اجراگر چیست؟
تمرین 4:کدام یک از موارد زیر شامل «کد ماشین» است؟
الف) اجراگر پایتون
ب) کیبورد
ج) فایل منبع پایتون
د) یک سند پردازش کلمه
تمرین 5:چه مشکلی در کد زیر وجود دارد؟

```
>>> primt ‘Hello world!’
File "<stdin>", line 1
  primt ‘Hello world!’
                      ^
SyntaxError: invalid syntax
>>>
```

تمرین 6: متغیری همانند x در کدام قسمت کامپیوتر ذخیره می شود پس از اینکه پایتون خط آن را تمام می کند؟

```
X = 123
```

الف) واحد پردازش مرکزی

ب) حافظهٔ اصلی
ج) حافظهٔ ثانویه
د) دستگاه های خروجی

تمرین 7: برنامهٔ زیر چه چیزی را پرینت می کند؟

```
X = 43
X = x - 1
print(x)
```

الف) 43
ب) 42
ج) x + 1
د) خطا می دهد چون x = x + 1 از نظر ریاضی غیرممکن است.

تمرین 8: هر یک از موارد بعدی را با استفاده از مثالی از توانایی های انسان توضیح دهید: (1) واحد پردازش مرکزی، (2) حافظهٔ اصلی، (3) حافظهٔ ثانویه، (4) دستگاه ورودی و (5) دستگاه خروجی. برای مثال، «معادل انسانی واحد پردازش مرکزی چیست»؟

فصل 2

متغیر ها، عبارات و دستورات

2.1 ارزش و نوع

مقدار (value) یکی از ساده‌ترین چیزهایی است که برنامه با آن کار می کند مثل یک حرف یا عدد. ارزش هایی که ما تا الان دیده‌ایم 1 و 2 و "Hello World!" هستند.
این مقدار هایی که نام بردیم انواع مختلفی هستند: 2 عددی صحیح است و "Hello World!" یک رشته متن یا همان String است. شما (و مترجم) می توانید این رشته متن ها را تشخیص دهید چون داخل علامت های نقل‌قول قرار دارند.
دستور پرینت برای اعداد صحیح هم کار می کند. ما با فرمان پایتون مترجم را باز می کنیم.

```
Python
>>> print(4)
4
```

اگر شما بتوانید به نوع مقدار پی ببرید، مترجم می تواند به شما بگوید.

```
>>> type('Hello, World!')
<class 'str'>
>>> type(17)
<class 'int'>
```

واضح است که رشته متن ها متعلق به نوع str هستند و اعداد صحیح یا همان integers متعلق به int هستند. قابل توجه است که اعداد اعشاری متعلق به نوع float هستند. چون این اعداد به شکلی نشان داده می شود که به آنها floating point گفته می شود.

```
>>> type(3.2)
<class 'float'>
```

پس اعدادی مانند "17" یا "3.2" چی؟ آنها هم اعداد هستند ولی داخل اعلامت های نقل قول قرار دارند.

```
>>> type('17')
<class 'str'>
>>> type('3.2')
<class 'str'>
```

رشته متن یا همان string هستند.

وقتی شما یک عدد صحیح تایپ می کنید شاید تمایل داشته باشید که بین سه هر دستهٔ سه رقمی از ویرگول استفاده کنید؛ مانند 1،000،000. این در پایتون یک عدد صحیح قانونی نیست، چیزی که قانونی است این است:

```
>>> print(1,000,000)
1 0 0
```

خب، این اصلاً چیزی که شما انتظارش را داشتید نیست! پایتون 1،000،000 را عدد صحیحی که با ویرگول جدا شده‌اند می بیند.
این اولین مثالی از خطای معنایی است که میبینیم: کد بدون ارسال پیام خطا اجرا می شود ولی کار درست را انجام نمی دهد.

2.2 متغیر ها

یکی از قوی‌ترین خاصیت های یک زبان برنامه‌نویسی، توانایی آن به تغییر متغیرها است. *متغیر (variable)* اسمی است که به یک مقدار داده می شود.
دستور انتسابی (assignment statement) متغیر های جدید می سازد و به آنها مقدار میدهد:

```
>>> message = 'And now for something completely different'
>>> n = 17
>>> pi = 3.1415926535897931
```

این مثال سه دستور انتسابی جدید می سازد. اولی متنی را به یک متغیر جدید وامی گذارد که اسم آن massage است؛ دومی عدد صحیح 17 را به n وامیگذارد؛ سومی مقدار تقریبی عدد پی را به pi وامی گذارد.
برای نشان دادن مقدار یک متغیر، می توانید از دستور پرینت استفاده کنید:

```
>>> print(n)
17
>>> print(pi)
3.141592653589793
```

منظور از نوع (type) یک متغیر، نوع مقداری است که به آن مربوط است.

```
>>> type(message)
<class 'str'>
>>> type(n)
<class 'int'>
>>> type(pi)
<class 'float'>
```

2.3 اسم متغیر ها و کلیدواژه ها

برنامه‌نویس ها به طور کلی اسم هایی را برای متغیر های خود انتخاب می کنند که معنادار باشند و کاربرد آن را واضح سازد. اسم متغیر ها می توانند به طور خودسرانه بلند و طولانی باشند. آنها می توانند هم شامل حروف باشند و هم شامل اعداد باشند ولی نمی توانند با یک عدد شروع شوند. شما نباید از حروف بزرگ انگلیسی استفاده کنید، این کار غیرقانونی است بهتر است اسم متغیرها را با حروف کوچک انگلیسی شروع کنید (بعداً میبینید چرا).

علامت زیرخط یا همان underscore (_) هم میتواند در اسم متغیر ظاهر شود. آن بیشتر در اسم هایی که شامل چند کلمه هستند استفاده می شود مانند my_name یا airspeed_of_unladen_swallow. نام متغیرها میتوانند با زیرخط شروع شود، اما ما اکثراً سعی می کنیم از آن استفاده نکنیم مگر اینکه در حال نوشتن کد کتابخانه باشیم.
اگر به متغیری اسم غیرقانونی و غیرقابل استفاده بدهید، خطای سینتاکس میگیرید:

```
>>> 76trombones = 'big parade'
SyntaxError: invalid syntax
>>> more@ = 1000000
SyntaxError: invalid syntax
 >>> class = 'Advanced Theoretical Zymurgy'
SyntaxError: invalid syntax
```

76trombones غیرقانونی است و شما اجازه استفاده از آن را ندارید چون با عدد شروع می شود. more@ غیرقانونی است چون شامل یک علامت غیرقانونی است، @. ولی چه مشکلی با کلاس وجود دارد؟
کلاس یکی از کلیدواژه های پایتون است. مترجم از کلیدواژه ها استفاده میکند تا ساختار برنامه را شناسایی کند و از آنها نمیتوانید به عنوان اسم متغیرها استفاده کنید.
پایتون 35 کلیدواژه رزرو شده دارد.

```
False      await      else       import     pass
None       break      except     in         raise
True       class      finally    is         return
and        continue   for        lambda     try
as         def        from       nonlocal   while
assert     del        global     not        with
async      elif       if         or         yield
```

شاید بخواهید این لیست را زیر دست داشته باشید. اگر مترجم درمورد نام یکی از متغیرهایتان شکایتی داشته باشد و شما دلیل آن را ندانید، به این لیست نگاه کنید.

2.4 عبارات اجرایی

یک *عبارت اجرایی* دسته‌ای از کد است که مترجم پایتون می تواند آن را اجرا کند. ما دو نوع عبارت اجرایی مشاهده کردیم: print که عبارت دستوری است و دستور انتسابی.
وقتی شما یک عبارت را در حالت تعاملی تایپ کنید، مترجم آن را اجرا می کند و نتیجه را اگر وجود داشته باشد نشان می دهد.
یک اسکریپت معمولاً شامل دنباله‌ای از عبارات است. اگر بیشتر از یکی وجود داشته داشته باشد، نتیجه ها یکی‌یکی با اجرای عبارات ظاهر می شوند.
برای مثال، این متن

```
print(1)
x= 2
print(x)
```

به شکل زیر ظاهر می شود:

عبارت دستور انتسابی هیچ خروجی حاصلی ندارد.

2.5 اوپراتور ها و اوپروند ها (عملگر ها و عملوند ها)

اوپراتور ها علامت های خاصی هستند که نشان دهندهٔ محاساباتی مانند جمع و ضرب هستند. مقادیری که اوپراتورها به آنها اعمال شده‌اند اوپراند ها (عملوندها) نامیده می شوند.
این اوپراتور ها +،-،*،/ و ** هستند که جمع، تفریق، ضرب، تقسیم و توان را نشان می دهند که در مثال های زیر به کار رفته‌اند:

```
20+32
hour-1
hour*60+minute
minute/60
5**2
(5+9)*(15-7)
```

تغییراتی در اوپرتور تقسیم میان پایتون 2 و پایتون 3 ایجاد شده است. در پایتون 3، نتیجهٔ تقسیم عددی اعشاری است:

```
>>> minute = 59
>>> minute/60
0.9833333333333333
```

اوپراتور تقسیم در پایتون 2 دو عدد صحیح را تقسیم می کرد و جواب را به یک عدد صحیح می برید:

```
>>> minute = 59
>>> minute/60
0
```

برای به دست آوردن همان نتیجه در پایتون 3 از تقسیم شکست دادن استفاده کنید (عدد صحیح//).

```
>>> minute = 59
>>> minute//60
0
```

در پایتون 3 تقسیم صحیح بیشتر شبیه زمانی است که عبارت را در ماشین حساب وارد کنید.

2.6 عبارات

یک عبارت ترکیب مقادیر، متغیرها و اوپراتور است. یک مقدار به تنهایی یک عبارت محسوب می شود، متغیر هم همینطور، پس عبارات زیر عبارات غیرقانونی هستند(با فرض اینکه مقداری به متغیر x واگذار شده است):

```
17
x
```

```
x + 17
```

اگر شما عبارتی را در حالت تعاملی وارد کنید، مترجم آن را *بررسی* می کند و نتیجه را نشان می دهد:

```
>>> 1 + 1
2
```

ولی در یک اسکریپت، یک عبارت به تنهایی نمیتواند کاری انجام دهد! این یکی از موضوعاتی است که موجب تعجب افراد مبتدی می شود.

تمرین 1: دستورات زیر را در مترجم پایتون تایپ کنید و ببینید چه اتفاقی می افتد:

```
5
x = 5
x + 1
```

2.7 ترتیب اوپراتور ها

وقتی بیشتر از یک اوپراتور در عبارتی به کار میرود، ترتیب بررسی به قوانین تقدم بستگی دارد. برای اوپراتور های ریاضیاتی، پایتون از قراردادهای ریاضی استفاده می کند و به آنها پایبند است. کلمهٔ PEMDAS روشی است برای به یاد آوردن این قوانین است:

- پرانتزها (parentheses) بر همگی تقدم دارد و می تواند برای بالابردن اولویت یک عبارت در ترتیبی که بخواهید استفاده شود. از آنجا که عبارات داخل پرانتز ها اول بررسی می شوند، (1-3)*2، 4 می شود و (5-2)**(1+1) هم 8 می شود. شما همچنین می توانید از پرانتز ها برای راحت کردن خواندن عبارات استفاده کنید، مانند (minute * 10) / 60، با اینکه جواب را تغییر نمی دهد.
- توان، جایگاه بعدی را از نظر تقدم دارد، پس 1+1**2 می شود 3 نه 4، و 3**1*3 هم 3 می شود نه 27.
- ضرب و تقسیم در یک اولویت قرار دارند که بالاتر از جمع و تفریق هستند که آنها هم در یک جایگاه و مرتبه قرار دارند. پس 1-3*2، 5 نمی شود 4 است و 6+4/2 هم 8 است نه 5.
- اوپراتورهایی که در یک اولویت قرار دارند از چپ به راست بررسی می شوند. پس عبارت 1-3-5، 1 است نه 3 چون 3-5 اول حساب می شود و بعد 1 از 2 تفریق می شود.

زمانی که شک می کنید، همیشه از پرانتز استفاده کنید تا مطمئن شوید که محاسابات به درستی و با همان ترتیبی که شما می خواهید انجام بگیرد.

2.8 اوپراتور باقی‌مانده

این اوپراتور با اعداد صحیح کار می کند و باقیمانده را زمانی به دست می آورد که اوپراند اول با دومی تقسیم می شود.
در پایتون، این اوپراتور با علامت درصد نشان داده می شود (%). سینتاکس آن مشابه اوپراتور های دیگر است:

```
>>> quotient = 7 // 3
>>> print(quotient)
2
```

```
>>> remainder = 7 % 3
>>> print(remainder)
1
```

پس 7 تقسیم بر 3 شده 2 است و 1 باقی‌مانده.
این اوپراتو به طور قابل تعجبی پرکاربرد است. برای مثال، می توانید بررسی کنید که یک عدد بخش‌پذیر بر عدد دیگری است: اگر y % x صفر باشد، پس x بر y بخش‌پذیر است.

2.9 محاسابات رشته متنی

اوپراتور + با رشته متن ها هم کار می کند ولی جمع ریاضیاتی نیست. در عوض آنها را به هم می چسباند که به معنای پیوستن رشته ها از طریق اتصال آنها به صورت پشت‌سرهم است. به عنوان نمونه:

```
>>> first = 10
>>> second = 15
>>> print(first+second)
25
>>> first = '100'
>>> second = '150'
>>> print(first + second)
100150
```

اوپراتور * هم با رشته‌متن ها با ضرب محتوای رشته‌متنی در عدد صحیحی ضرب می شود. برای نمونه:

```
>>> first = 'Test'
>>> second = 3
>>> print(first * second)
Test Test Test
```

2.10 درخواست ورودی از کاربر

گاهی ما ترجیح می دهیم مقدار متغیر را با استفاده از کیبورد از کاربر بگیریم. پایتون یک تابع در ساختار خود به اسم **input** دارد که از کیبورد ورودی می گیرد. وقتی از این تابع استفاده شود، برنامه توقف می کند و منتظر می ماند تا کاربر چیزی را تایپ کنید. زمانی که کاربر **Return** یا **Enter** را بزند، این برنامه ادامه می دهد و **input** این ورودی را به عنوان رشته متن برمیگرداند.

```
>>> inp = input()
Some silly stuff
>>> print(inp)
Some silly stuff
```

قبل از اینکه از کاربر درخواست ورودی کنید، ایدهٔ خوبی است که پیامی پرینت کنید که کاربر چه چیزی وارد کند.
می‌توانید یک رشته (string) را به تابع **input** بدهید تا پیش از توقف برای دریافت ورودی، به کاربر نمایش داده شود:

```
>>> name = input('What is your name?\n')
What is your name?
Chuck
>>> print(name)
```

Chuck

نوشتهٔ \n در انتهای پیام ورودی (prompt) نشان دهندهٔ یک *خط جدید* است، این یک نویسهٔ خاص است که باعث شکستن خط (line break) می شود. به همین سبب ورودی کاربر پایین پیام ورودی وارد می شود.
اگر بخواهید که کاربر عددی صحیح تایپ کند، شما می توانید مقدار وارد شده را با استفاده از تابع int() به int تبدیل کند:

```
>>> prompt = 'What...is the airspeed velocity of an unladen swallow?\n'
>>> speed = input(prompt)
What...is the airspeed velocity of an unladen swallow?
17
>>> int(speed)
17
>>> int(speed) + 5
22
```

اگر کاربر چیزی غیر از رشته‌ای از اعداد تایپ کند، خطا نشان می دهد:

```
>>> speed = input(prompt)
What...is the airspeed velocity of an unladen swallow?
What do you mean, an African or a European swallow?
>>> int(speed)
ValueError: invalid literal for int() with
base 10: 'What do you mean, an African or a European swallow?'
```

بعداً میبینیم که چطور اینگونه خطاها را حل کنیم.

2.11 کامنت ها

با بزرگتر و پیچیده‌تر شدن برنامه ها، خواندن آنها سخت تر می شود. زبان های رسمی متراکم هستند و معمولاً کار سختی است به تکه‌ای از کد نگاه کنید تا بفهمید چه اتفاقی می افتد یا چرا.
به همین دلیل، اضافه کردن یادداشت ها به برنامه‌هایتان فکر خوبی است تا در زبان طبیعی توضیح دهید که برنامه چه کاری می کند.
این یادداشت ها کامنت نام دارند و در پایتون با علامت # آغاز می شود:

```
# compute the percentage of the hour that has elapsed
percentage = (minute * 100) / 60
```

در این نمونه، کامنت در یک خط نوشته شده است. شما همچنین می توانید کامنت ها را در انتهای یک خط بنویسید:

```
percentage = (minute * 100) / 60     # percentage of an hour
```

از هر چیزی بعد # صرف نظر می شود؛ هیچ اثری روی برنامه ندارد.
کامنت ها زمانی بیشترین کابرد را دارند که ویژگی های غیرواضح کد را ثبت کنند. با این کار، منطقی است که فرض کنیم خواننده بفهمد کد *چه کاری* می کند؛ اما بهتر است که توضیح دهیم *چرا* این کار را انجام می دهد.
کامنت زیر با توجه به کد تکراری و بی‌فایده است:

```
v = 5       #assign 5 to v
```

کامنت زیر شامل اطلاعات مفید و باارزش است که در خود کد آورده نشده:

```
v = 5       #velocity in meters/second.
```

انتخاب اسم های مناسب برای متغیر ها می تواند شما را از استفاده از کامنت ها بی نیاز کند، اما اسم های طولانی می تواند خواندن عبارات را پیچیده سازد پس اینجا یک بده‌بستان وجود دارد.

2.12 انتخاب نام‌های حافظه‌یار برای متغیر ها

تا زمانی که قانون های ساده برای نام‌گذاری متغیرها را دنبال کنید و پیرو آنها باشید و از استفاده از کلمات رزرو شده خودداری کنید، اختیار زیادی برای انتخاب اسم متغیرهایتان دارید. در ابتدا، شاید زمانی که برنامه‌ای را بخوانید و برنامه های خودتان بنویسید این اختیار انتخاب گیج‌کننده باشد. برای نمونه، سه برنامهٔ زیر کاربرد یکسانی دارند ولی وقتی که میخواهید آنها را بخوانید و سعی کنید آن را بفهمید متفاوت‌اند.

```
a = 35.0
b = 12.50
c = a * b
print(c)
```

```
hours = 35.0
rate = 12.50
pay = hours * rate
print(pay)
```

```
x1q3z9ahd = 35.0
x1q3z9afd = 12.50
x1q3p9afd = x1q3z9ahd * x1q3z9afd
print(x1q3p9afd)
```

مترجم پایتون همهٔ این برنامه ها را *دقیقاً عین هم* می بیند ولی انسان ها اینها را متفاوت می بینند. انسان ها هدف برنامهٔ دوم را سریع می فهمند چون برنامه‌نویس اسم هایی را برای متغیر ها انتخاب کرده است که هدف هر یک از آنها را از لحاظ اینکه کدام داده ها در هر یک از متغیرها ذخیره می شود مشخص می کند.
ما به این متغیرهایی که با دقت زیاد نامیده شده‌اند «نام‌های حافظه‌یار متغیر» می گوییم. از این اسم ها استفاده می کنیم تا به ما کمک کند به یاد بیاوریم چرا این متغیر را ساخته‌ایم.
درحالی که همهٔ اینها خوب به نظر می رسند و استفاده از اسم‌های حافظه‌یار برای متغیرها فکر خوبی است، این اسم‌ها میتوانند کار برنامه‌نویس های مبتدی را در فهمیدن برنامه سخت کند. زیرا برنامه‌نویس های مبتدی کلمه های رزرو شده را حفظ نکرده‌اند (فقط 35 تا از آنها وجود دارد) و گاهی نام‌هایی که بیش از حد توصیفی هستند، شروع می کنند شبیه بخشی از خود زبان شوند، نه صرفاً نام‌هایی که به خوبی انتخاب شده‌اند.
نگاهی سریع به نمونه کد پایتون بیندازید که روی برخی داده ها حلقه می زند(تکرار انجام می دهد). به زودی در فصل های آینده با حلقه ها آشنا خواهیم شد، اما اکنون سعی کنید معنای کد زیر را درک کنید:

```
for word in words:
      print(word)
```

اینجا چه اتفاقی می افتد؟ کدام یک از نشانه ها (for, word, in, etc.) کلمات رزرو شده هستند و کدام فقط اسم متغیرها هستند؟ آیا پایتون در سطح بنیادی مفهوم کلمات را درک می‌کند؟ برنامه‌نویس های مبتدی در درک اینکه کدام بخش‌های این کد *باید* همانند این کد باشند و کدام بخش ها انتخابات برنامه‌نویس است دچار مشکل می شوند.
کد زیر با کد بالایی یکسان است:

```
for slice in pizza:
     print(slice)
```

برای یک برنامه‌نویس مبتدی نگاه کردن به این کد و درک اینکه کدام بخش‌ها کلمات رزرو شده از طرف پایتون و کدام اسم‌هایی برای متغیرها هستند که از طرف برنامه‌نویس انتخاب شده‌اند آسان تر است. به خوبی واضح است که پایتون هیچ اطلاعات پیشین از پیتزا و لقمه های آن ندارد و نمیداند پیتزا از یک یا چند لقمه ساخته شده است.
ولی اگر برنامهٔ ما واقعاً درمورد خواندن داده ها و جستوجوی کلماتی در داده است، `pizza` و `slice` برای این برنامه اسم هایی هستند که یادآورد هیچ‌کدام از داده ها نیستند. انتخاب آنها به عنوان اسم داده هایمان ما را از معنی واقعی برنامه غافل می سازد.
بعد از مدتی کوتاه، شما با پرکاربردترین اسم های رزرو شده آشنایی پیدا خواهید کرد و کلمات رزرو شده بیشتر به چشمتان میخورند:
بخش هایی از کد که از طرف پایتون تعریف شده‌اند (`for, in, print, and :`) پررنگ شده‌اند و اسم‌های منتخب برنامه‌نویس(`word, words`) پررنگ نشده‌اند. بسیاری از ویرایشگرهای متن با سینتاکس‌های پایتون آگاه هستند و کلمات رزرو شده را متفاوت رنگ می کنند تا به شما نشانه هایی بدهد تا فرق بین متغیرها و کلمات رزرو شده مشخص باشد. بعد از مدتی شما شروع به خواندن پایتون می کنید و سریع می فهمید کدام متغیرند و کدام کلمات رزرو شده هستند.

2.13 دیباگ کردن

در این نقطه، تنها خطای سینتاکسی که شما میتوانید دریافت کنید این است که اسم غیرقانونی برای متغیر خود انتخاب کنید، مانند `class` و `yield` که کلیدواژه هستند یا `odd~job` و `US$` که شامل کاراکترهای غیرقانونی هستند.
اگر بین کاراکترها در نام‌های متغیر فاصله وجود داشته باشد، پایتون فکر می کند که دو عملوند (operands) هستند که اوپراتور ندارند:

```
>>> bad name = 5
SyntaxError: invalid syntax
```

برای خطاهای سینتاکس، پیام‌های خطا کمک زیادی نمی کنند. پیام های پرتکرار به شکل `SyntaxError: invalid syntax` اطلاعات زیادی به ما نمی دهند.
خطای زمان اجرایی(runtime error) که بیشتر ممکن است مواجهه آن شوید، "use before def;" است که سعی دارد متغیری را قبل از اینکه مقداری به آن بدهید استفاده کند. این زمانی اتفاق می افتد که در نوشتن اسم متغیران تفاوت املائی وجود داشته باشد:

```
>>> principal = 327.68
```

```
>>> interest = principle * rate
NameError: name 'principle' is not defined
```

نام متغیر ها کیس سنسیتیو هستند یعنی به بزرگی یا کوچکی حروف حساس‌اند، پس `LaTeX` با `latex` یکی نیست.
در این حالت، به احتمال زیاد دلیل هرگونه خطای معنایی ترتیب اوپراتورها است. برای نمونه، برای اینکه مقدار $1/2\pi$ را بررسی کنید، شاید تمایل داشته باشید اینگونه بنویسید:

```
>>> 1.0 / 2.0 * pi
```

تقسیم اول اتفاق می افتد، پس عدد $\pi/2$ را به دست می آورید که با هم یکی نیستند! هیچ راهی وجود ندارد که پایتون حدس بزند شما چه می خواهید بنویسید، پس در این حالت شما پیام خطا نمیگیرید؛ بلکه جوابی غلط به دست می آورید.

2.14 واژه‌نامه

دستور انتسابی (assignment) مقداری را به متغیری واگذاری می کند.
به هم چسباندن(concatenate) دو اوپراند یا همان عملوند را به هم چسباندن
کامنت(comment) اطلاعاتی در یک برنامه که برای بقیه برنامه‌نویس ها است (یا هر کسی که کد منبع را می خواند) و هیچ تاثیری در اجرای برنامه ندارد.
بررسی (evaluate) ساده‌سازی یک دستور با اجرای هر یک از عملیات به ترتیب برای به دست آوردن یک مقدار مشخص.
عبارات (Expression) ترکیبی از متغیرها، اوپراتور ها و مقادیری که نشان‌دهندهٔ یک مقدار نهایی هستند.
عدد اعشاری (floating point) عددی که نشان‌دهندهٔ اعدادی با واحد اعشاری هست.
عدد صحیح (integer) نوعی عدد که نشان‌دهندهٔ اعداد کامل است.
کلیدواژه(keyword) کلمهٔ رزرو شده‌ای که از طرف کامپایلر برای تجزیهٔ برنامه استفاده می شود؛ شما نمی توانید از کلیدواژه هایی مانند `if, def, while` به عنوان اسم متغیر استفاده کنید.
حافظه‌یار (mnemonic) ما اکثراً به متغیرها اسم های حافظه‌یار می دهیم تا به ما کمک کنند آنچه را در آن ذخیره کرده‌ایم، به یاد بیاوریم.
اوپراتور باقی‌مانده (modulus operator) اوپراتوری که با علامت درصد `(%)` نشان داده شده است، روی اعداد صحیح کار می کنند و باقی‌مانده را زمانی که یک عدد بر دیگری تقسیم می شود.
اوپراند یا عملوند (operand) یکی از مقادیری که اوپراتور روی آن کار می کند.
اوپراتور (operator) یک علامت خاص که نشان‌دهندهٔ محاسبهٔ ساده‌ای است مانند جمع، ضرب یا به هم چسباندن رشته متن ها.
قوانین ترتیب محاسباتی (rules of precedence) دسته‌ای از قوانین که ترتیب بررسی دستورات دارای اوپراتورها و اوپراندها را کنترل می کنند.
دستورات (statement) بخشی از کد که نشان‌دهندهٔ دستور و فرمان است. تا اینجا، دستوراتی که مشاهده کرده‌ایم دستورات انتسابی وعبارت دستور پرینت هستند.
رشته متن (String) نویسه‌ای که نشان دهندهٔ دسته‌ای از کاراکتر ها و حروف است.
تایپ (type) گروهی از مقادیر. تایپ هایی که ما تا اینجا با آنها آشنا شده‌ایم اعداد صحیح (type `int`)، اعداد اعشاری (type `float`) و رشته متن (type `str`).
مقدار(value) دسته‌ای ساده از داده ، مانند یک عدد یا متن که برنامه‌ای کنترل می کند.
متغیر(variable) اسمی که به یک مقدار اشاره دارد.

2.15 تمرین ها

تمرین 2: برنامه‌ای بنویسید که از `input` استفاده می کند تا یک کاربر را هدایت کند که اسمش را بنویسد و به آنها سلام دهد و خوش آمدگویی کند.

```
Enter your name: Chuck
Hello Chuck
```

تمرین 3: برنامه‌ای بنویسید که کاربر را هدایت کند تا ساعت و درآمد ساعتی خود را وارد کند و درآمد کامل آن را حساب کند.

```
Enter Hours: 35
Enter Rate: 2.75
Pay: 96.25
```

فعلاً نیاز نیست که نگران داشتن دقیقاً دو عدد اعشار باشیم. اگر می خواهید، می توانید از تابع ساختاری پایتون `round` استفاده کنید تا از آن عدد تقریب بگیرید.

تمرین 4: فرض کنید ما دستور های انتسابی زیر را اجرا کرده‌ایم:

```
width = 17
height = 12.0
```

برای هر یک از عبارت های زیر، مقدار عبارت را بنویسید و تایپ مقدار آن را بنویسید.

```
1. width//2
2. width/2.0
3. height/3
4. 1 + 2 * 5
```

از مترجم پایتون برای بررسی جواب هایتان استفاده کنید.

تمرین 5: برنامه‌ای بنویسید که کاربر را به وارد کردن دمایی در سلسیوس هدایت کند، این دما را به فارنهایت تبدیل کند و دمای تبدیل شده را تایپ کند.