

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 کارهایمان را برای ما انجام دهد. نرم افزار های امروزه به طوری ساخته شده اند 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 سخت افزار 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 حافظه عظیمی هستند و می توانند برای ما خیلی سودمند باشند اگر ما 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 خسته کننده ای 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» باید مهارت پیدا کنید. هنگامی که شما این زبان جدید را یاد بگیرید، شما می توانید که این فعالیت روزمره و تکراری را به عهده همکاران بگذارید (همان کامپیوتر) و برای انجام کار هایی 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 و علاقه 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 شما یک نرم افزاری 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 شما 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 موبایلرآن را جدا کنید، موارد زیر را داخل آن پیدا می کنید:

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

- وسیله های ورودی و خروجی، به طور 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. به عبارتی، این شبکه، شکل آهسته تر و بعضی مواقع غیر قابل اعتماد تر از حافظه ثانویه است.

با اینکه اکثر این نکات ریز که این در زمینه ها کار می کنند را بهتر است به عهده سازنده های کامپیوتر بگذاریم. این به ما کمک بسیاری می کند که واژه نامه ای داشته باشیم که در مورد این بخش های مختلف در عین نوشتن برنامه حرف بزنیم.

1.4: تو کجا هستی؟

به عنوان یک برنامه نویس، وظیفه شما استفاده و تنظیم هر یک از این منابع است تا مسئله ای 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 گوئید که بعداً چه کاری را انجام دهد. گاهی شما به CPU می گوئید که از حافظه اصلی، حافظه ثانویه، شبکه، یا وسایل ورودی و خروجی استفاده کنید.

شما باید همان کسی باشید که به سوال «کار بعدی چیست؟» پاسخ دهد. اما اگر شما را فقط 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 دستور اتان را از قبل بنویسید. ما به این اطلاعاتی که از پیش نوشته می شوند، برنامه می گوئیم و عمل نوشتن این راهنمایی ها و به خصوص درست نوشتن آنها برنامه نویسی نام دارد.

1.3 درک برنامه نویسی

در ادامه این کتاب، ما 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 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 می‌دهیم. ما با خواندن و توصیف برنامه شروع میکنیم، بعد برنامه های ساده می‌نویسیم، در نهایت به تدریج شروع به نوشتن برنامه های سخت‌تر و پیچیده‌تر شروع می‌کنیم. در برهه‌ای از زمان شما الهام خود را می‌یابید و الگوها را به تنهایی پیدا می‌کنید و بیشتر به طور طبیعی مشاهده می‌کنید که چگونه یک مسئله را درک کنید و برنامه ای برای حل آن بنویسید. وقتی به آن سطح می‌رسید، برنامه نویسی به فرایندی لذت‌بخش و خلاقانه تبدیل می‌شود.

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

1.4 کلمات و جملات

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

وقتی ما یک سگ را تربیت می‌کنیم، ما از کلمات خاصی مانند «بشین»، «بایست» و «بیار» استفاده می‌کنیم. هنگامی که شما با یک سگ صحبت می‌کنید و از هیچ یک از کلمات خاص استفاده نمی‌کنید، آنها فقط با یک نگاه پیچیده به شما نگاه می‌کنند تا اینکه از یک کلمه خاص استفاده کنید. به عنوان مثال، اگر شما بگویید: «ای کاش افراد بیشتری برای بهبود سلامت خود پیاده روی می‌کردند»، چیزی که بیشتر سگ ها به احتمال زیاد می‌شنوند این است، «فلان فلان پیاده روی فلان فلان فلان». این به همان دلیل است که «پیاده روی» یک کلمه 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 می‌کنند شما 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 می‌گویید «try»، پایتون هر دفعه که شما می‌گویید بدون اشتباه امتحان می‌کند.

ما درباره این کلمات رزرو شده و راه استفاده از آنها در زمان مناسب یاد می‌گیریم، اما اکنون ما بر روی معادل «صحبت کردن» (در زبان انسان به سگ) برای پایتون تمرکز می‌کنیم. نکته ی خوب در مورد فرمان دادن به پایتون که حرف بزنند این است که ما حتی می‌توانیم با قرار دادن پیاممان داخل علامت های نقل قول (""):

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

الان ما حتی اولین جمله صحیح از نظر سینتاتیک خود را نوشته ایم. جمله ی 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 نقل قول تکی استفاده می کنند، مگر در مواردی که نقل قول 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 مکالمه ای با پایتون را آغاز کنیم تا مهارت های زبانی خود را امتحان کنیم.

قبل از اینکه بتوانید با پایتون صحبت کنید، شما باید ابتدا نرم افزار پایتون را دانلود کنید و یاد بگیرید که چگونه پایتون را در 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 داده ام. در مقطعی، شما وارد یک ترمینال یا پنجره دستورات می شوید و واژه `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 `>>>` شیوه مترجم پایتون برای پرسیدن این سوال از شماست: «می خواهی در مرحله ی بعد چه کاری انجام دهی؟» 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 ساده ترین کلمات و جملات زبان پایتون را نمی دانید. شما شاید تمایل داشته باشید که این جمله ی استاندارد و معروفی که فضانوردان هنگام فرود روی سیاره ای دور دست 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 درباره ی 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 آورده اید و حالا به همین صفحه رجوع می کنی و دوباره تلاش می کنی:

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

Hello world!

الان این خیلی بهتر است، پس شما 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 نحوه نگارش حساس است، اما پایتون هوشمند نیست. در واقع شما فقط دارید با خودتان حرف می زنید ولی با نگارش درست. به عبارتی، زمانی که شما از یک برنامه ای استفاده می کنید که توسط شخصی دیگر نوشته شده است، مکالمه بین شما و برنامه نویس 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 برنامه نویس های خواهید بود که از پایتون برای صحبت کردن با کاربر برنامه خودتان به کار می برید. قبل از اینکه اولین مکالمه خود را با مترجم پایتون به پایان برسانیم، بهتر است راه مناسب «خداحافظ» گفتن را هنگام ارتباط با ساکنان سیاره 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 کلمه ی 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 احتمالاً 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, و 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:

```
001010001110100100101010000001111
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، برنامه نویس های کمی با زبان ماشین می نویسند. در عوض، ما 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 سخت افزار کامپیوتر مرتبط استفاده است، زبان ماشین قابل حمل بین سخت افزار های مختلف نیست. برنامه هایی که با زبان های سطح بالا نوشته می شوند را می توان بین کامپیوتر های مختلف جابه جا کرد؛ این کار با استفاده از یک مترجم متفاوت روی دستگاه جدید یا با کامپایلر مجدد کد برای ایجاد نسخه ای 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) کامپایلر ها.

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 مقداری 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
```


وقتی می‌خواهیم یک برنامه بنویسیم از یک ویرایشگر متن استفاده می‌کنیم تا فرمان‌های پایتون را در فایلی بنویسیم که به آن اسکریپت گفته می‌شود. در نتیجه، اسکریپت‌های پایتون با اسم `.py` تمام می‌شوند.

برای اجرای اسکریپت، شما باید به مترجم پایتون اسم فایل را بگویید. در یک پنجره 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 «\$» نشاندهنده اعلان سیستم‌عامل است و دستور «`cat hello.py`» به ما نشان می‌دهد که فایل «`hello.py`» شامل برنامه‌تک خطی پایتون برای یک رشته است. ما مترجم پایتون را فعال می‌کنیم و دستور می‌دهیم که کد منبع فایل «`hello.py`» را بخواند به جای اینکه منتظر بماند تا خطوط کد پایتون را یکی‌یکی از ما دریافت کند.

متوجه می‌شوید که نیازی به استفاده از `quit()` در انتهای فایل برنامه پایتون نبود. زمانی که پایتون کد منبع شما را از یک فایل می‌خواند، می‌داند زمانی که به انتهای فایل می‌رسد باید توقف کند.

1.8 برنامه چیست؟

تعریف یک برنامه در ساده‌ترین حالت، دنباله‌ای از دستورات پایتون است که با دقت طراحی شده هستند که یک کاری را انجام دهند. حتی اسکریپت ساده `hello.py` یک برنامه است. یک برنامه‌تک خطی است اگرچه چنان کاربردی نیست، اما در دقیق‌ترین تعریف، این یک برنامه‌ی پایتون است.

به احتمال زیاد راحت‌ترین راه فهمیدن این که یک برنامه چیست، این است که اول به یک مسئله فکر کنیم که یک برنامه می‌تواند برای حل آن ساخته شود و بعد به برنامه‌ای نگاه کنیم که آن مسئله را حل می‌کند.

فرض کنیم شما درباره محاسبات اجتماعی از روی پست‌های فیس‌بوک (Facebook) تحقیق می‌کنید و شما می‌خواهید پرتکرارترین کلمه در سری‌های مختلف پست‌ها را بدانید. شما برای این کار می‌توانید همه آن پست‌ها را پرینت کنید و متن همه آنها را یکی‌یکی بررسی کنید تا پرتکرارترین کلمه را پیدا کنید، اما این کار زمان بسیار زیادی می‌کشد و احتمال خطای بسیار بالایی دارد. کار عاقلانه این می‌باشد که شما یک برنامه پایتون بنویسید تا بتوانید این کار را به سرعت زیاد و دقیق انجام دهید و آخر هفته خود را مشغول به تفریح باشید.

برای مثال، به متن زیر درباره یک دلک 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 میلیون‌ها نوشته هستید. در حقیقت، یاد گرفتن پایتون و نوشتن یک برنامه پایتون برای شمردن این کلمات سریع‌تر از انجام این کار به صورت دستی است.

خبر بهتر این است که من هم‌اکنون یک برنامه‌ای نوشتم تا پرتکرارترین کلمه را در فایل نوشته بباید. آن را نوشتم، امتحان کردم و الان هم آن را در اختیار شما قرار می‌دهم تا شما 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
```

شما حتی لازم نیست به پایتون تسلط داشته باشید تا از این برنامه استفاده کنید. شما باید فصل 10 این کتاب را بخوانید که تمام تکنیک‌هایی که در نوشتن این برنامه به کار رفته را به طور کامل بفهمید. شما کاربر نهایی این برنامه هستید شما فقط از برنامه استفاده می‌کنید و در برابر قدرت و هوش آن شگفت‌زده می‌شوید که چگونه شما را از انجام کاری خسته کننده نجات داد. شما فقط این کد را داخل یک فایل به نام words.py و آن را اجرا می‌کنید 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 می‌کند. پایتون راهی مناسب برای تبادل دنباله‌هایی از دستورات مفید (یعنی برنامه‌ها) در یک زبان مشترک و قابل استفاده برای هر کسی که برنامه پایتون را در کامپیوتر خود نصب کند. پس هیچ یک از ما در حال صحبت با پایتون نیستیم، در واقع ما از طریق پایتون با هم حرف می‌زنیم.

1.9 اجزای سازنده برنامه‌ها

در چند فصل آینده، ما بیشتر دربارهٔ واژه‌نامه، ساختار جمله، ساختار پاراگراف و ساختار داستان پایتون یاد می‌گیریم. ما دربارهٔ توانایی‌های شگفت‌آور پایتون یاد می‌گیریم و چگونه این توانایی‌ها را در کنار هم قرار دهیم تا برنامه‌های مفید بسازیم.

الگوهای مفهومی ساده‌ای وجود دارد که ما از آن برای ساختن و طراحی برنامه‌ها استفاده می‌کنیم. این ساختارها فقط برای برنامه‌های پایتون نیستند، اینها همه بخشی از هر زبان برنامه‌نویسی هستند؛ از زبان ماشین گرفته تا زبان‌های سطح بالا.

ورودی از «دنیای خارجی» داده جمع‌آوری کنید. این کار می‌تواند بررسی داده‌ها و اطلاعات از یک فایل باشد یا حتی نوعی سنسور مانند میکروفون یا جی‌پی‌اس. در برنامه‌های اولیه‌ما، ورودی‌ما از طریق داده‌تایپ شده از طرف کاربر با کیبورد به دست می‌آید.

خروجی نتیجه‌های 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 فرمان‌ها را یک بار بنویسید و به آنها اسمی دهید، سپس در صورت نیاز، آن دستورات را در بخش‌های مختلف برنامه‌ی خود دوباره استفاده کنید.

این موضوع تقریباً بیش از حد آسان به نظر می‌رسد 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 کاربر‌های آن بسازد.

برنامه‌شمردن کلمات بالا به طور مستقیم از همه‌این‌الگوها 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 می‌شود پایتون بررسی ادامه‌ی برنامه‌ی شما را متوقف کند.

برنامه نویسان مبتدی از این نکته که پایتون هیچ جایی برای اشتباه نمیدهد این برداشت را می‌کنند که پایتون سخت‌گیر، بی‌رحم و خشن است. درحالی‌که پایتون از دیگران خوشش می‌آید و با آنها خوب است، آنها فکر می‌کنند پایتون آنها را شخصاً می‌شناسد و از آنها کینه به دل دارد. به خاطر این کینه، پایتون برنامه‌هایی که به نظر ما کاملاً صحیح نوشته شده‌اند، به عنوان «نامناسب» رد می‌کند فقط برای اینکه می‌خواهد ما را اذیت کند.

```
>>> print 'Hello world!'
File "<stdin>", line 1
      print 'Hello world!'
      ~~~~~
```

SyntaxError: invalid syntax

```
>>> print ('Hello world')
Traceback (most recent call last):
  File "<stdin>", line 1, in
```

NameError: name 'print' 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 دعا کردن با پایتون به دست نمی آید. پایتون فقط به وسیله است. هیچ احساساتی ندارد و خوشحال و حاضر این است که به فرمان و دستورات شما عمل کند. پیام های ارور آن سخت گیر و بی رحم به نظر می رسند ولی آنها فقط فریاد های پایتون برای کمک هستند. به چیزهایی که شما نوشته اید نگاه کرده است و درک نمی کند شما چه چیزی نوشته اید.

پایتون بیشتر شبیه یک سگ است، شما را بی نهایت دوست دارد، چند کلیدواژه ای دارد که شما را درک کند، با نگاهی 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 به چیزی به من میگی ولی من آن را نمیفهمم ولی لطفاً با من حرف بزن (>>)».

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

خطا های سینتاکس (Syntax errors) اینها اولین خطاهایی هستند که با آنها مواجه می شوید و همچنین راحت ترین آنها برای رفع هستند. یک خطای سینتاکس به این معنی است که شما قوانین «قواعد» پایتون را زیر پا گذاشته اید. پایتون تلاش زیادی می کند که خط و کاراکتری که خطای شما را تشخیص داده را به طور دقیق نشان دهد. تنها نکته ای قابل توجه درباره خطای سینتاکس این است که گاهی خطایی که نیاز به تصحیح است قبل تر از آن است که پایتون نشان می دهد و متوجه شده است که متن را نمیفهمد. پس خط و کاراکتری که پایتون نشان می دهد میتواند نقطه شروع بررسی شما باشد.

خطا های منطقی (Logic errors) یک خطای منطقی زمانی است که برنامه شما سینتاکس خوبی دارد ولی خطایی در ترتیب دستورات یا شاید خطایی در چگونگی ارتباط دستورات شما به یکدیگر وجود دارد. نمونه ای 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 در بطری را ببند.»

خطا های معنایی (Semantic 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 گوئید: «...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 مزرعه ای هستند و پشت یک طویله راه می روند و هیچ رستورانی نمی بینند. بعد شما می گوئید: «در پمپ گاز به چپ بپیچیدی یا راست؟» و بعد می گویند: «من آدرسی که تو دادی را به طور دقیق دنبال کردم، آنها را نوشتم، نوشته ام در پمپ گاز به چپ بپیچ و یک مایل جلو برو.» سپس شما می گوئید: «من خیلی متأسفم، چون با اینکه راهنمایی های من از نظر سینتاتیک کاملاً درست است ولی متأسفانه یک خطای 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 داده اید 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 را ابتدا کنید، دوباره به خودتان بخوانید، بعد 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 که ساختار اولیه را بسازید.

تفکر عمیق کمی فکر کنید! چه نوع خطایی است: سینتکس، 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) سخت 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 به کار بیاید وقتی مشکل برنامه خطای تایپی است، ولی اگر مشکل مفهومی است این روش بی فایده است. اگر شما نفهمید که خطای شما چه کاری می کند، 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، شما 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 درباره اینکه مشکل چیست نیاز دارید. اگر دو یا چند احتمال وجود دارد، سعی کنید درباره 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 سؤالتان 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 می‌کنند چون تحمل نوشتن دوباره برنامه را ندارد (حتی اگر غلط باشد). اگر راضی شوید حتی می‌توانید برنامه را قبل از اینکه پاک کنید، آن را به فایل دیگری کپی کنید. می‌توانید تکه‌های آن را یکی‌یکی اضافه کنید.

1.12 روند یادگیری

در صورتی که در ادامه کتاب پیش می‌روید، اگر موضوعات در نگاه اول با هم بی‌ربط به نظر برسند نترسید. وقتی شما اول حرف زدن را یاد می‌گرفتید، ایرادی نداشت که شما در چند سال اول حرف زدن صدا های بامزه‌ای میدادید و ایرادی نداشت اگر شش ماه طول کشید که کلمات ساده به جملات ساده تبدیل شود و 5 یا 6 سال طول کشید که این جملات به پاراگراف‌ها تبدیل شد و چند سال دیگر تا بتوانید به تنهایی یک داستان کوتاه بنویسید.

ما میخواهیم شما پایتون را با سرعتی بیشتر یاد بگیرید، به همین سبب همه را در کنار هم در طول چند فصل آینده یاد می‌دهیم. اما این روند همانند یاد گرفتن یک زبان جدید زمان می‌برد که آن را هضم و درک کنید و بعد به طور طبیعی آن را درک کنید. این موضوع باعث کمی تعجب می‌شود، چرا که ما مرتب به مباحث سر می‌زنیم و دوباره به آنها برمی‌گردیم تا هم‌زمان که در حال تعریف بخش‌های کوچک هستیم، شما بتوانید تصویر کلی را هم ببینید. درحالی که این کتاب به صورت خطی نوشته شده است و اگر در دوره‌ای شرکت می‌کنید آن هم به صورت خطی پیش می‌رود، شما می‌توانید 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، شما می‌توانید دلیل برنامه‌نویسی را بهتر بفهمید. با مرور مطالب قبلی و حتی حل دوباره مسائل قبلی؛ پی می‌برید که بسیاری از مطالبی که قبلاً خواندید را می‌فهمید و آنها را درک می‌کنید حتی اگر مطالبی که در حال حاضر به آنها خیره شده‌اید غیرقابل درک به نظر برسند.

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 می‌کوبیدید بلند می‌کنید و یک قدم عقب‌تر می‌روید و متوجه می‌شوید که واقعاً در حال ساختن یک مجسمه‌ی زیبا هستید.

اگر چیزی به‌ویژه سخت به نظر می‌رسد، معمولاً فایده‌ای ندارد که تمام شب را بیدار بمانید و خیره 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 می‌دهم 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 یک خطا در یک برنامه

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

کامپایل (compile) به عمل ترجمه 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 فرمان‌ها و عبارات بعد از دیدن پرامپت (نشانه ورودی).

تفسیر و اجرا کردن (interpret) اجرای برنامه‌ای در زبانی بالاسطح با ترجمه آن 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 می‌شود.

حافظه اصلی (main memory) برنامه‌ها و داده‌ها را ذخیره می‌کند. حافظه اصلی تمام اطلاعاتش را پس از خاموش شدن از دست می‌دهد.

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 نشانه ورودی (prompt) زمانی که یک برنامه پیامی را نشان می‌دهد و توقف می‌کند که کاربر به آن ورودی تایپ کند.

حافظه ثانویه (secondary memory) برنامه‌ها و داده‌ها را ذخیره می‌کند و اطلاعات آن را حفظ می‌کند حتی زمانی که خاموش است. به طور کلی کندتر از حافظه اصلی است. مثال‌هایی از حافظه ثانویه شامل درایوهای دیسک و حافظه فلش در فلش‌مموری‌های USB.

سمانتیک‌ها یا همان معناشناسی (semantic) معنی یک برنامه

خطای معنایی خطایی در یک برنامه که باعث می‌شود کاری را خارج از آنچه برنامه‌نویس منظور داشت انجام دهد.

کد منبع برنامه‌ای در زبان سطح بالا.

1.14 تمرین‌ها

تمرین 1: کاربرد حافظه ثانویه در یک کامپیوتر چیست؟

(الف) تمام محاسبات و منطق برنامه را اجرا کند.

(ب) حفظ کردن صفحات وب در اینترنت.

(ج) ذخیره 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؟

```
>>> print 'Hello world!'
File "<stdin>", line 1
    print 'Hello world!'
    ^
```

SyntaxError: invalid syntax

```
>>>
```

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

`X = 123`

(الف) واحد پردازش مرکزی

(ب) حافظه اصلی

(ج) حافظه ثانویه

(د) دستگاه های خروجی

تمرین 7: برنامه 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) حافظه اصلی، (3) حافظه ثانویه، (4) دستگاه ورودی و (5) دستگاه خروجی. برای مثال، «معادل انسانی واحد

پردازش مرکزی چیست»؟