

Herkes İçin Python

Python 3 ile Veri Keşfi

Dr. Charles R. Severance

Türkçe Çeviri
Dr. Altay Arbak

Bölüm 1

Neden program yazmayı öğrenmelisiniz?

Program yazmak ya da programlama (programming), son derece yaratıcı ve insana doyum veren bir etkinliktir. Program yazmak için pek çok nedeniniz olabilir: geçiminizi sağlamaktan zorlu bir veri analizi problemini çözmeye, eğlenmekten başka birinin bir problemini çözmesine yardım etmeye kadar. Bu kitap, *herkesin* programlamayı bilmesi gerektiğini ve programlamayı öğrendiğinizde yeni edindiğiniz becerilerle ne yapmak istediğinizi kendiniz keşfedeceğinizi varsayıyor.

Günlük hayatımızda dizüstü bilgisayarlardan cep telefonlarına kadar pek çok bilgisayarla çevriliyiz. Bu bilgisayarları, bizim adımıza pek çok işi halledebilen “kişisel asistanlarımız” olarak düşünebiliriz. Günümüz bilgisayarlarındaki donanım (hardware), temelde bize sürekli olarak şu soruyu soracak şekilde tasarlanmıştır: “Şimdi ne yapmamı istersiniz?”

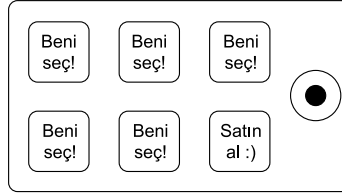
Programcılar donanımın üzerine bir işletim sistemi (operating system) ve bir dizi uygulama ekler; sonuçta da pek çok farklı işi yapmamıza yardım edebilen, oldukça kullanışlı ve becerikli bir Kişisel Dijital Asistan elde ederiz.

Bilgisayarlarımız hızlıdır ve muazzam miktarda belleğe sahiptir; bilgisayara ondan “sırada ne yapmasını” istediğimizi anlatacak dili bir bilsek, bize çok yardımcı olabilirler. Bu dili bilseydik, tekrar gerektiren işleri bizim yerimize yapmasını bilgisayara söyleyebilirdik. İşin ilginç yanı, bilgisayarların en iyi yaptığı işler çoğu zaman biz insanların sıkıcı ve beyin uyuşturucu bulduğu türden işlerdir.

Örneğin, bu bölümün ilk üç paragrafına bakın ve bana en sık kullanılan kelimeyi ve



Şekil 1.1: Kişisel Dijital Asistan



Şekil 1.2: Size Seslenen Programcılar

bu kelimenin kaç kez kullanıldığını söyleyin. Kelimeleri birkaç saniye içinde okuyup anlayabilmiş olsanız da onları saymak neredeyse eziyettir, çünkü bu, insan zihninin çözmek için tasarlandığı türden bir problem değildir. Bilgisayar için durum tam tersidir: Bir kâğıttaki metni okuyup anlamak bilgisayar için zordur, ama kelimeleri sayıp en çok kullanılan kelimenin kaç kez kullanıldığını size söylemek bilgisayar için çok kolaydır:

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

“Kişisel bilgi analizi asistanımız”, bu bölümün ilk üç paragrafında “to” kelimesinin on altı kez kullanıldığını hemen söyledi. (Ç.N.: Sayım, bölümün İngilizce aslındaki ilk üç paragraf üzerinden yapılmıştır.)

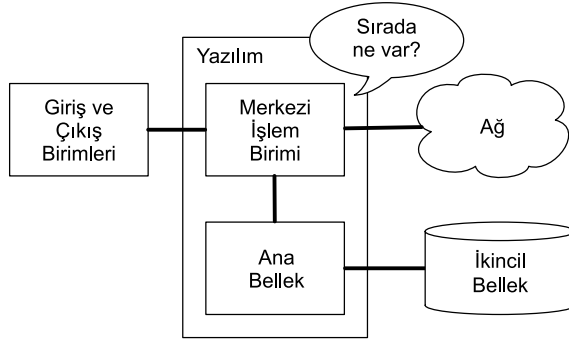
İşte bilgisayarların, insanların iyi olmadığı işlerde iyi olması, “bilgisayar dili” konuşmakta ustalaşmanız gerekmesinin asıl nedenidir. Bu yeni dili öğrendiğinizde, sıradan ve sıkıcı işleri ortağınıza (bilgisayara) devredebilir, size özgü yeteneklerinizle yapabileceğiniz işlere daha çok zaman ayırabilirsiniz. Bu ortaklığa yaratıcılığı, sezgiyi ve yenilikçiliği siz getirirsiniz.

1.1 Yaratıcılık ve motivasyon

Bu kitap profesyonel programcılar için yazılmamış olsa da profesyonel programcılık hem maddi hem de manevi açıdan son derece tatmin edici bir meslek olabilir. Başkalarının kullanacağı faydalı, zarif ve akıllıca programlar yapmak çok yaratıcı bir etkinliktir. Bilgisayarınızda ya da Kişisel Dijital Asistanınızda (PDA) genellikle farklı programcı gruplarının yazdığı pek çok program bulunur ve bunların her biri dikkatinizi ve ilginizi çekmek için yarışır. Programcılar ihtiyaçlarınızı karşılamak ve bu arada size harika bir kullanıcı deneyimi sunmak için ellerinden geleni yaparlar. Bazı durumlarda, bir yazılımı (software) seçtiğinizde programcılar sırf bu seçiminiz sayesinde doğrudan para kazanır.

Programları, programcı gruplarının yaratıcı ürünleri olarak düşünürsek, belki de aşağıdaki şekil Kişisel Dijital Asistanımızı daha doğru anlatıyordur:

Şimdilik asıl motivasyonumuz para kazanmak ya da son kullanıcıları (end users) memnun etmek değil; hayatımızda karşılaştığımız veri ve bilgilerle uğraşırken daha verimli olmak. İlk başladığınızda, yazdığınız programların hem programcısı hem de son kullanıcısı siz olacaksınız. Programcı olarak beceri kazandıkça ve programlama size daha yaratıcı gelmeye başladıkça, aklınız başkaları için program geliştirmeye kayabilir.



Şekil 1.3: Donanım Mimarisi

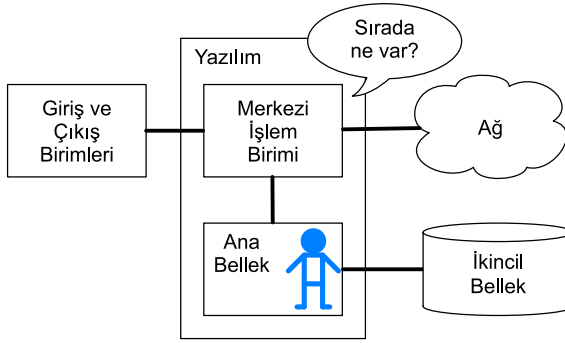
1.2 Bilgisayar donanım mimarisi

Yazılım geliştirmek için bilgisayarlara talimat verirken kullanacağımız dili öğrenmeye başlamadan önce, bilgisayarların nasıl yapıldığını biraz öğrenmemiz gerekiyor. Bilgisayarınızı ya da cep telefonunuzu söküp içine derinlemesine bakacak olsaydınız, şu parçaları bulurdunuz:

Bu parçaların genel tanımları şöyledir:

- *Merkezi İşlem Birimi* (Central Processing Unit ya da kısaca CPU), bilgisayarın “sırada ne var?” sorusuyla yatıp kalkacak şekilde yapılmış parçasıdır. Bilgisayarınızın hızı 3,0 gigahertz ise, CPU saniyede üç milyar kez “Sırada ne var?” diye soruyor demektir. CPU’ya yetişebilmek için çok hızlı konuşmayı öğrenmeniz gerekecek.
- *Ana bellek* (main memory), CPU’nun acilen ihtiyaç duyduğu bilgileri saklamak için kullanılır. Ana bellek neredeyse CPU kadar hızlıdır. Ama ana bellekte saklanan bilgiler, bilgisayar kapatıldığında yok olur.
- *İkincil bellek* (secondary memory) de bilgi saklamak için kullanılır, ama ana bellekten çok daha yavaştır. İkincil belleğin avantajı, bilgisayara elektrik gelmediğinde bile bilgiyi saklayabilmesidir. Disk sürücüler ya da flaş bellekler (genellikle USB belleklerde ve taşınabilir müzik çalarlarda bulunur) ikincil bellek örnekleridir.
- *Giriş ve çıkış birimleri* (input and output devices) basitçe ekranımız, klavyemiz, faremiz, mikrofonumuz, hoparlörümüz, dokunmatik yüzeyimiz gibi parçalardır. Bilgisayarla etkileşim kurmanın bütün yolları bunlardır.
- Günümüzde çoğu bilgisayarda, bir ağ üzerinden bilgi almak için bir de *ağ bağlantısı* (network connection) bulunur. Ağı, her zaman “ayakta” olmayabilen ve veri saklayıp geri almak için kullanılan çok yavaş bir yer olarak düşünebiliriz. Yani bir bakıma ağ, *ikincil belleğin* daha yavaş ve zaman zaman güvenilmez bir biçimidir.

Bu bileşenlerin nasıl çalıştığına dair ayrıntıların çoğunu bilgisayar üreticilerine bırakmak en iyisi olsa da, program yazarken bu farklı parçalardan söz edebilmek için biraz terminoloji bilmek işimize yarar.



Şekil 1.4: Siz Neredesiniz?

Bir programcı olarak işiniz, çözmeniz gereken problemi çözmek ve bu çözümden elde ettiğiniz verileri analiz etmek için bu kaynakların her birini kullanmak ve bir orkestra şefi gibi yönetmektir. Programcı olarak çoğunlukla CPU ile “konuşacak” ve ona sırada ne yapacağını söyleyeceksiniz. Bazen de CPU’ya ana belleği, ikincil belleği, ağı ya da giriş/çıkış birimlerini kullanmasını söyleyeceksiniz.

CPU’nun “Sırada ne var?” sorusunu yanıtlayan kişi siz olmalısınız. Ama sırf saniyede üç milyar kez komut verebilirsiniz diye sizi 5 mm boyuna küçültüp bilgisayarın içine yerleştirmek hiç de rahat olmazdı. Bu yüzden talimatlarınızı önceden yazmanız gerekir. Saklanan bu talimatlara *program*, bu talimatları yazma ve doğru hâle getirme işine de *programlama* diyoruz.

1.3 Programlamayı anlamak

Kitabın geri kalanında sizi programlama sanatında ustalaşmış biri hâline getirmeye çalışacağız. Sonunda bir *programcı* olacaksınız; belki profesyonel bir programcı değil, ama en azından bir veri/bilgi analizi problemine bakıp o problemi çözecek bir program geliştirme becerisine sahip olacaksınız.

Bir bakıma, programcı olmak için iki beceriye ihtiyacınız var:

- Birincisi, programlama dilini (Python) bilmeniz gerekir; yani kelime dağarcığını ve dilbilgisini bilmelisiniz. Bu yeni dildeki kelimeleri doğru yazabilmeli ve bu dilde kurallara uygun “cümleler” kurmayı bilmelisiniz.
- İkincisi, “bir hikâye anlatmanız” gerekir. Hikâye yazarken, okura bir fikri aktarmak için kelimeleri ve cümleleri bir araya getirirsiniz. Hikâye kurmanın bir becerisi ve sanatı vardır; hikâye yazma becerisi de yazdıkça ve geri bildirim aldıkça gelişir. Programlamada programımız “hikâye”, çözmeye çalıştığınız problem ise “fikir”dir.

Python gibi bir programlama dilini öğrendikten sonra, JavaScript ya da C++ gibi ikinci bir programlama dilini öğrenmenin çok daha kolay olduğunu göreceksiniz. Yeni programlama dilinin kelime dağarcığı ve dilbilgisi çok farklıdır, ama problem çözme becerileri bütün programlama dillerinde aynıdır.

Python'un "kelimelerini" ve "cümlelerini" oldukça çabuk öğreneceksiniz. Yepyeni bir problemi çözecek tutarlı bir program yazabilmeniz ise daha uzun sürecek. Programlamayı, yazı yazmayı öğrettiğimiz gibi öğretiyoruz. Önce programları okuyup açıklayarak başlıyoruz, sonra basit programlar yazıyoruz, zamanla da giderek daha karmaşık programlar yazıyoruz. Bir noktada "ilham perileriniz gelir", kalıpları kendi başınıza görmeye başlarsınız ve bir problemi ele alıp onu çözen bir programı nasıl yazacağınızı daha doğal bir şekilde görürsünüz. O noktaya geldiğinizde de programlama çok keyifli ve yaratıcı bir sürece dönüşür.

Python programlarının kelime dağarcığı ve yapısıyla başlıyoruz. Basit örnekler size okumayı ilk öğrendiğiniz günleri hatırlatacak; sabırlı olun.

1.4 Kelimeler ve cümleler

İnsan dillerinin aksine, Python'un kelime dağarcığı aslında oldukça küçüktür. Bu "kelime dağarcığına" "ayrılmış kelimeler" (reserved words) ya da "anahtar kelimeler" (keywords) diyoruz. Bunlar Python için çok özel anlamı olan kelimelerdir. Python bu kelimeleri bir Python programında gördüğünde, bu kelimelerin Python için tek bir anlamı vardır, başka hiçbir anlamı yoktur. İlerleyen bölümlerde program yazarken, sizin için anlamı olan kendi kelimelerinizi de uyduracaksınız; bunlara *değişken* (variable) denir. Değişkenlerinize ad seçerken oldukça serbest olacaksınız, ama Python'un ayrılmış kelimelerinin hiçbirini değişken adı olarak kullanamazsınız.

Bir köpeği eğitirken "otur", "bekle" ve "getir" gibi özel kelimeler kullanırız. Bir köpekle konuşurken bu ayrılmış kelimelerden hiçbirini kullanmazsanız, siz bir ayrılmış kelime söyleyene kadar yüzünde şaşkın bir ifadeyle size bakar. Örneğin, "Keşke daha çok insan genel sağlığını iyileştirmek için yürüyüşe çıksa" dersiniz, çoğu köpeğin muhtemelen duyduğu şey şudur: "bla bla bla *yürüyüş* bla bla bla bla." Bunun nedeni, "yürüyüş" kelimesinin köpek dilinde ayrılmış bir kelime olmasıdır. Pek çok kişi, insanlarla kediler arasındaki dilde hiç ayrılmış kelime olmadığını söyleyebilir¹.

İnsanların Python'la konuştuğu dildeki ayrılmış kelimeler şunlardır:

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

Hepsi bu kadar. Üstelik köpeğin aksine Python zaten tamamen eğitilmiş durumdadır. "try" (dene) dediğinizde, Python bunu her söyleyişinizde şaşmadan dener.

Bu ayrılmış kelimeleri ve nasıl kullanıldıklarını zamanı geldikçe öğreneceğiz, ama şimdilik (insan-köpek dilindeki) "konuş" komutunun Python'daki karşılığına odaklanacağız. Python'a konuşmasını söylemenin güzel yanı, tırnak içinde bir mesaj vererek ona ne söyleyeceğini bile söyleyebilmemizdir:

¹<http://xkcd.com/231/>

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

Böylece sözdizimi açısından doğru ilk Python cümlemizi de yazmış olduk. Cümlemiz *print* fonksiyonuyla (function) başlıyor, ardından da tek tırnak içine alınmış, bizim seçtiğimiz bir metin (string) geliyor. *print* deyimlerindeki (statement) metinler tırnak içine alınır. Tek tırnak ve çift tırnak aynı işi görür; çoğu kişi tek tırnak kullanır, ancak metnin içinde tek tırnak (aynı zamanda kesme işareti) geçiyorsa çift tırnağı tercih eder.

1.5 Python’la sohbet

Artık Python’da bildiğimiz bir kelime ve basit bir cümle olduğuna göre, yeni dil becerilerimizi sınamak için Python’la nasıl sohbet başlatacağımızı bilmemiz gerekiyor.

Python’la sohbet edebilmek için önce Python yazılımını bilgisayarınıza kurmanız ve Python’ı bilgisayarınızda nasıl başlatacağınızı öğrenmeniz gerekir. Bu konu bu bölüm için fazla ayrıntılı; bu yüzden www.py4e.com adresine bakmanızı öneririm. Orada Python’ı Macintosh ve Windows sistemlerinde kurmak ve başlatmak için ayrıntılı talimatlar ve ekran kaydı videoları hazırladım. Bir noktada bir terminalde ya da komut penceresinde *python* yazacaksınız; Python yorumlayıcısı (interpreter) da etkileşimli modda (interactive mode) çalışmaya başlayacak ve aşağı yukarı şöyle görünecek:

```
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.
>>>
```

>>> istemi (prompt), Python yorumlayıcısının size “Şimdi ne yapmamı istersiniz?” diye sorma biçimidir. Python sizinle sohbet etmeye hazır. Tek bilmeniz gereken, Python dilini nasıl konuşacağınızdır.

Diyelim ki en basit Python kelimelerini ya da cümlelerini bile bilmiyorsunuz. Uzak bir gezegene inen ve gezegenin sakinleriyle konuşmaya çalışan astronotların kullandığı o klasik cümleyi denemek isteyebilirsiniz:

```
>>> 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
>>>
```

Bu pek iyi gitmedi. Hızlıca bir şey düşünmezseniz, gezegenin sakinleri muhtemelen sizi mızraklarıyla şişleyecek, bir şişe geçirecek, ateşte kızartacak ve akşam yemeğinde yiyecekler.

Neyse ki yolculuğa çıkarken bu kitabın bir kopyasını yanınıza almışsınız; hemen bu sayfayı açıp yeniden deniyorsunuz:

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

Bu çok daha iyi görünüyor, siz de iletişimi biraz daha ilerletmeyi deniyorsunuz:

```
>>> 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)
>>>
```

Sohbet bir süre çok iyi gidiyordu, sonra Python dilini kullanırken en ufak bir hata yaptınız ve Python mızrakları yeniden çıkardı.

Bu noktada şunu da fark etmelisiniz: Python inanılmaz derecede karmaşık ve güçlü olsa da, onunla iletişim kurarken kullandığınız sözdizimi (syntax) konusunda çok titiz olsa da, Python akıllı *değildir*. Aslında yalnızca kendinizle sohbet ediyorsunuz, sadece doğru sözdizimini kullanarak.

Bir bakıma, başka birinin yazdığı bir programı kullandığınızda sohbet, sizinle o programcılar arasında geçer; Python da aracılık eder. Python, programları yazanların sohbetin nasıl ilerlemesi gerektiğini ifade etmelerinin bir yoludur. Yalnızca birkaç bölüm sonra, siz de Python'ı kullanarak programınızın kullanıcılarıyla konuşan o programcılardan biri olacaksınız.

Python yorumlayıcısıyla ilk sohbetimizden ayrılmadan önce, Python Gezegeni'nin sakinleriyle konuşurken “hoşça kal” (good-bye) demenin doğru yolunu da bilerseniz iyi olur:

```
>>> 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()
```

İlk iki yanlış denemede hatanın farklı olduğunu fark edeceksiniz. İkinci hata farklıdır, çünkü *if* ayrılmış bir kelimedir; Python bu ayrılmış kelimeyi gördü ve bir şey söylemeye çalıştığımızı, ama cümlenin sözdizimini yanlış kurduğumuzu düşündü.

Python'a "hoşça kal" demenin doğru yolu, etkileşimli >>> istemine *quit()* yazmaktır. Bunu tahmin etmeniz muhtemelen epey uzun sürerdi; bu yüzden elinizin altında bir kitap bulunması büyük ihtimalle işinize yarayacak.

(Ç.N.: Bu bölümdeki örnek diyaloglar, kodun kendisi değiştirilmediği için İngilizce kalmıştır. Türkçe karşılıkları şöyledir: "I come in peace, please take me to your leader" → "Barış içinde geldik, lütfen bizi liderinize götürün"; "You must be the legendary god that comes from the sky" → "Gökten inen efsanevi tanrı siz olmalısınız"; "We have been waiting for you for a long time" → "Sizi uzun zamandır bekliyorduk"; "Our legend says you will be very tasty with mustard" → "Efsanemize göre hardalla çok lezzetli olacaksınız"; "We will have a feast tonight unless you say" → "Bu gece ziyafet çekeceğiz, tabii siz şunu demezseniz"; "if you don't mind, I need to leave" → "Sakıncası yoksa gitmem gerekiyor".)

1.6 Terminoloji: Yorumlayıcı ve derleyici

Python, insanların okuması ve yazması, bilgisayarların da okuması ve işlemesi görece kolay olacak şekilde tasarlanmış *yüksek seviyeli* (high-level) bir dildir. Diğer yüksek seviyeli diller arasında Java, C++, PHP, Ruby, Basic, Perl, JavaScript ve daha pek çoğu sayılabilir. Merkezi İşlem Biriminin (CPU) içindeki asıl donanım bu yüksek seviyeli dillerin hiçbirini anlamaz.

CPU, *makine dili* (machine language) dediğimiz bir dili anlar. Makine dili çok basittir ve açıkçası yazması çok yorucudur, çünkü tamamen sıfırlar ve birlerle ifade edilir:

```
001010001110100100101010000001111
11100110000011101010010101101101
...
```

Yalnızca sıfırlar ve birlerden oluştuğu için makine dili ilk bakışta oldukça basit görünür, ama sözdizimi Python'unkinden bile daha karmaşık ve çok daha girift yapıdadır. Bu yüzden makine dilinde program yazan programcı sayısı çok azdır. Bunun yerine, programcıların Python ya da JavaScript gibi yüksek seviyeli dillerde yazabilmesini sağlayan çeşitli çeviriciler (translator) geliştiririz; bu çeviriciler de programları, CPU'nun asıl yürütebilmesi için makine diline dönüştürür.

Makine dili bilgisayar donanımına bağlı olduğu için farklı donanım türleri arasında *taşınabilir* (portable) değildir. Yüksek seviyeli dillerde yazılmış programlar ise yeni makinede farklı bir yorumlayıcı kullanılarak ya da kod yeniden derlenerek programın yeni makineye uygun bir makine dili sürümü oluşturulduğunda farklı bilgisayarlara taşınabilir.

Bu programlama dili çeviricileri iki genel kategoriye ayrılır: (1) yorumlayıcılar ve (2) derleyiciler.

Bir *yorumlayıcı*, programın programcı tarafından yazılan kaynak kodunu (source code) okur, kaynak kodu ayrıştırır (parse) ve talimatları anında yorumlar. Python bir yorumlayıcıdır; Python'ı etkileşimli olarak çalıştırdığımızda bir satır Python (bir cümle) yazabiliriz, Python da bu satırı hemen işler ve bir sonraki Python satırını yazmamız için hazır hâle gelir.

Bazı Python satırları, Python'a bir değeri (value) daha sonra kullanmak üzere hatırlamasını istediğinizi söyler. Hatırlanacak bu değer için bir ad seçmemiz gerekir; değeri daha sonra geri almak için de bu sembolik adı kullanabiliriz. Saklanan bu verilere erişmek için kullandığımız etiketleri *değişken* terimiyle adlandırırız.

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

Bu örnekte Python'dan altı değerini hatırlamasını ve değeri daha sonra geri alabilmemiz için x etiketini kullanmasını istiyoruz. Python'un değeri gerçekten hatırladığını *print* ile doğruluyoruz. Sonra Python'dan x 'i alıp yediyle çarpmasını ve yeni hesaplanan değeri y 'ye koymasını istiyoruz. Ardından Python'dan y 'de o anda bulunan değeri yazdırmasını istiyoruz.

Bu komutları Python'a birer satır hâlinde yazıyor olsak da, Python bunları sıralı bir deyimler dizisi olarak ele alır; sonraki deyimler önceki deyimlerde oluşturulan verilere erişebilir. Böylece dört cümleden oluşan, mantıklı ve anlamlı bir sırayla dizilmiş ilk basit paragrafımızı yazmış oluyoruz.

Yukarıda gösterildiği gibi etkileşimli bir sohbet yürütebilmek, *yorumlayıcının* doğasında vardır. *Derleyiciye* (compiler) ise programın tamamını bir dosya içinde vermek gerekir; derleyici bu dosyayı alıp yüksek seviyeli kaynak kodu makine diline çeviren bir süreç çalıştırır, ardından elde edilen makine dilini daha sonra yürütül-
mek üzere bir dosyaya kaydeder.

Windows sistemi kullanıyorsanız, bu çalıştırılabilir (executable) makine dili programlarının dosya uzantısı çoğunlukla “.exe” ya da “.dll” olur; bunlar sırasıyla “executable” (çalıştırılabilir) ve “dynamic link library” (dinamik bağlantı kütüphanesi) anlamına gelir. Linux ve Macintosh’ta ise bir dosyayı tek başına çalıştırılabilir olarak işaretleyen bir uzantı yoktur.

Çalıştırılabilir bir dosyayı bir metin düzenleyicide açacak olsaydınız, dosya tamamen çılgınca ve okunamaz görünürdü:

```
?ELF^A^A^A^@^@^@^@^@^@^@^@^@B^@^C^@^A^@^@^@^@x82
^D^H4^@^@^@^@x90^]^@^@^@^@^@^@^@4^@^@^@C^@^@($^@!^@^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^H^Q^VhT^@x83^D^H^@xe8
.....
```

Makine dilini okumak ya da yazmak kolay değildir; bu yüzden Python ya da C gibi yüksek seviyeli dillerde yazmamızı sağlayan *yorumlayıcıların* ve *derleyicilerin* var olması güzel bir şey.

Derleyiciler ve yorumlayıcılar hakkındaki tartışmamızın bu noktasında, Python yorumlayıcısının kendisini de biraz merak ediyor olmalısınız. Hangi dilde yazılmış? Derlenen bir dilde mi yazılmış? “python” yazdığımızda tam olarak ne oluyor?

Python yorumlayıcısı “C” adlı yüksek seviyeli bir dilde yazılmıştır. Python yorumlayıcısının asıl kaynak koduna, www.python.org adresine gidip kaynak kodun bulunduğu yere ulaşarak bakabilirsiniz. Yani Python’un kendisi de bir programdır ve makine koduna (machine code) derlenmiştir. Python’ı bilgisayarınıza kurduğunuzda (ya da üretici kurduğunda), çevrilmiş Python programının makine kodu hâlindeki bir kopyasını sisteminize kopyalamış oldunuz. Windows’ta Python’un kendisine ait çalıştırılabilir makine kodu büyük ihtimalle şuna benzer bir adı olan bir dosyadadır:

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

Bu, bir Python programcısı olmak için gerçekten bilmeniz gerekenden fazlası; ama bazen insanın aklını kurcalayan bu küçük soruları daha en başta yanıtlamak iyidir.

1.7 Program yazmak

Python yorumlayıcısına komutlar yazmak, Python’un özelliklerini denemek için harika bir yoldur, ama daha karmaşık problemleri çözmek için önerilmez.

Bir program yazmak istediğimizde, Python talimatlarını bir metin düzenleyiciyle bir dosyaya yazarız; bu dosyaya *betik* (script) denir. Geleneksel olarak Python betiklerinin adları .py ile biter.

Betiği çalıştırmak için Python yorumlayıcısına dosyanın adını söylemeniz gerekir. Bir komut penceresinde aşağıdaki gibi `python hello.py` yazarsınız:

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

“\$” işletim sisteminin istemidir; “cat hello.py” ise bize “hello.py” dosyasında bir metni ekrana yazdıran tek satırlık bir Python programı olduğunu gösteriyor.

Python yorumlayıcısını çağırıyor ve bizden Python kodu satırlarını etkileşimli olarak istemek yerine, kaynak kodunu “hello.py” dosyasından okumasını söylüyoruz.

Dosyadaki Python programının sonuna *quit()* koymaya gerek olmadığını fark edeceksiniz. Python kaynak kodunuzu bir dosyadan okurken, dosyanın sonuna geldiğinde durması gerektiğini bilir.

1.8 Program nedir?

En temel tanımıyla bir *program*, bir iş yapmak üzere özenle hazırlanmış bir Python deyimleri dizisidir. Basit *hello.py* betiğimiz bile bir programdır. Tek satırlık bir programdır ve pek bir işe yaramaz, ama tanımın en katı hâliyle bir Python programıdır.

Programın ne olduğunu anlamamanın en kolay yolu, bir programın çözmek için yazılabileceği bir problemi düşünüp ardından o problemi çözecek bir programa bakmak olabilir.

Diyelim ki Facebook gönderileri üzerine bir sosyal bilişim (Social Computing) araştırması yapıyorsunuz ve bir dizi gönderide en sık kullanılan kelimeyi merak ediyorsunuz. Facebook gönderilerinin akışını yazıcıdan çıkarıp en yaygın kelimeyi bulmak için metnin üzerinde saatlerce kafa yorabilirsiniz, ama bu çok uzun sürer ve hataya çok açıktır. Bu işi hızlı ve doğru bir şekilde halletmek için bir Python programı yazmak akıllıca olur; böylece hafta sonunu eğlenceli bir şeyler yaparak geçirebilirsiniz.

Örneğin, bir palyaço ve bir araba hakkındaki aşağıdaki metne bakın. Metni inceleyin ve en yaygın kelimenin hangisi olduğunu ve kaç kez geçtiğini bulun.

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

Şimdi de bu işi milyonlarca satırlık metne bakarak yaptığınızı düşünün. Açıkçası, Python öğrenip kelimeleri sayan bir Python programı yazmanız, kelimeleri elle taramanızdan daha çabuk olurdu.

Daha da iyi haber şu: Bir metin dosyasındaki en yaygın kelimeyi bulan basit bir programı ben zaten yazdım. Yazdım, test ettim ve şimdi biraz zaman kazanabilmemiz için kullanımınıza sunuyorum.

```
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>

Bu programı kullanmak için Python bilmenize bile gerek yok. Programı yazarken kullanılan harika Python tekniklerini tam olarak anlamak için bu kitabın 10. bölümüne kadar gelmeniz gerekecek. Siz son kullanıcısınız; programı kullanırsınız, ne kadar akıllıca yazıldığına ve sizi ne kadar çok elle yapılacak işten kurtardığına hayran kalırsınız. Tek yapmanız gereken, kodu *words.py* adlı bir dosyaya yazıp

çalıştırmak ya da kaynak kodu <http://www.py4e.com/code3/> adresinden indirip çalıştırmaktır.

Bu, Python'un ve Python dilinin sizinle (son kullanıcı) benim (programcı) aramda nasıl bir aracı görevi gördüğünün güzel bir örneği. Python, bilgisayarına Python kuran herkesin kullanabileceği ortak bir dilde, işe yarar talimat dizilerini (yani programları) birbirimizle paylaşmamızın bir yoludur. Yani ikimiz de *Python*'la konuşmuyoruz; aslında Python *aracılığıyla* birbirimizle iletişim kuruyoruz.

1.9 Programların yapı taşları

Sonraki birkaç bölümde Python'un kelime dağarcığını, cümle yapısını, paragraf yapısını ve hikâye yapısını daha ayrıntılı öğreneceğiz. Python'un güçlü yeteneklerini ve işe yarar programlar oluşturmak için bu yetenekleri nasıl bir araya getireceğimizi öğreneceğiz.

Program oluştururken kullandığımız bazı düşük seviyeli (low-level) kavramsal kalıplar vardır. Bu yapılar yalnızca Python programlarına özgü değildir; makine dilinden yüksek seviyeli dillere kadar her programlama dilinin bir parçasıdır.

giriş (input) “Dış dünyadan” veri almak. Bu, bir dosyadan veri okumak, hatta mikrofona ya da GPS gibi bir tür algılayıcıdan veri almak olabilir. İlk programlarımızda girişimiz, kullanıcının klavyeden yazdığı verilerden gelecek.

çıkış (output) Programın sonuçlarını ekranda göstermek, bir dosyaya kaydetmek ya da müzik çalmak veya metni seslendirmek için hoparlör gibi bir cihaza göndermek.

sıralı yürütme (sequential execution) Deyimleri, betikte karşılaştıkları sırayla birbiri ardına gerçekleştirmek.

koşullu yürütme (conditional execution) Belirli koşulları kontrol etmek ve buna göre bir deyimler dizisini yürütmek ya da atlamak.

tekrarlı yürütme (repeated execution) Bir grup deyim, genellikle her seferinde biraz değiştirilerek, tekrar tekrar gerçekleştirmek.

yeniden kullanım (reuse) Bir grup talimatı bir kez yazıp onlara bir ad vermek, sonra bu talimatları programınızın her yerinde gerektiğinde yeniden kullanmak.

Kulağa neredeyse gerçek olamayacak kadar basit geliyor; elbette hiçbir zaman bu kadar basit değildir. Bu, yürümenin basitçe “bir ayağı diğerinin önüne koymak” olduğunu söylemeye benzer. Program yazma “sanatı”, bu temel öğeleri defalarca bir araya getirip birbirine örerek kullanıcılar için işe yarar bir şey ortaya çıkarmaktır.

Yukarıdaki kelime sayma programı, bu kalıplardan biri hariç hepsini doğrudan kullanıyor.

1.10 Ne ters gidebilir ki?

Python'la yaptığımız ilk sohbetlerde gördüğümüz gibi, Python kodu yazarken çok kesin bir dille iletişim kurmamız gerekir. En küçük bir sapma ya da hata, Python'un programınıza bakmaktan vazgeçmesine yol açar.

Programlamaya yeni başlayanlar, Python’un hataya hiç yer bırakmamasını çoğu zaman Python’un huysuz, nefret dolu ve zalim olduğunun kanıtı olarak görürler. Onlara göre Python başka herkesi seviyor gibi görünür, ama onları şahsen tanır ve onlara kin besler. Bu kin yüzünden de sırf bize eziyet etmek için kusursuz yazılmış programlarımızı alıp “uygun değil” diyerek reddeder.

```
>>> 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 <module>
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

>>>
```

Python’la tartışmanın pek bir faydası yok. O yalnızca bir araç. Duyguları yoktur; ne zaman ihtiyaç duyarsanız size hizmet etmeye mutlu ve hazırdır. Hata mesajları sert gelebilir, ama bunlar aslında Python’un yardım çağrısıdır. Yazdığınıza bakmıştır ve girdiğiniz şeyi bir türlü anlayamamaktadır.

Python daha çok bir köpeğe benzer: Sizi koşulsuz sever, anladığı birkaç anahtar kelimesi vardır, yüzünde tatlı bir ifadeyle size bakar (>>>) ve anlayacağı bir şey söylemenizi bekler. Python “SyntaxError: invalid syntax” dediğinde, aslında yalnızca kuyruğunu sallayıp şöyle diyor: “Bir şey söylediniz gibi geldi ama ne demek istediğinizi hiç anlamadım; yine de lütfen benimle konuşmaya devam edin (>>>).”

Programlarınız giderek daha gelişmiş hâle geldikçe, üç genel hata türüyle karşılaşacaksınız:

Sözdizimi hataları (syntax errors) Bunlar yapacağınız ilk hatalardır ve düzeltmesi en kolay olanlardır. Sözdizimi hatası, Python’un “dilbilgisi” kurallarını çiğnediğiniz anlamına gelir. Python, kafasının karıştığını fark ettiği satırı ve karakteri tam olarak göstermek için elinden geleni yapar. Sözdizimi hatalarının tek aldatici yanı, düzeltilmesi gereken hatanın bazen aslında programda Python’un kafasının karıştığını *fark ettiği* yerden daha önce olmasıdır. Yani Python’un bir sözdizimi hatasında gösterdiği satır ve karakter, araştırmanız için yalnızca bir başlangıç noktası olabilir.

Mantık hataları (logic errors) Mantık hatası, programınızın sözdizimi doğru olduğu hâlde deyimlerin sırasında ya da belki deyimlerin birbiriyle ilişkisinde bir yanlışlık olmasıdır. Mantık hatasına güzel bir örnek şu olabilir: “Su şişenizden bir yudum için, şişeyi sırt çantanıza koyun, kütüphaneye yürüyün, sonra da şişenin kapağını kapatın.”

Anlam hataları (semantic errors) Anlam hatası, atılacak adımları tarif ettiğinizin sözdizimi açısından kusursuz ve doğru sırada olduğu, ama programda yine de bir yanlışlık bulunduğu durumdur. Program tamamen doğrudur, ama sizin *amaçladığınız* şeyi yapmaz. Basit bir örnek vermek gerekirse, birine bir restoranın yolunu tarif ederken şöyle dediğinizi düşünün: “...benzin istasyonunun olduğu kavşağa gelince sola dön, bir buçuk kilometre git, restoran solunda kırmızı bir bina.” Arkadaşınız çok gecikir ve sizi arayıp bir çiftlikte, bir ahırın arkasında dolaştığını, ortada restorandan eser olmadığını söyler. Siz “benzin istasyonunda sola mı döndün, sağa mı?” diye sorarsınız, o da şöyle der: “Tarifine harfiyen uydum, not almıştım; benzin istasyonunda sola dön ve bir buçuk kilometre git yazıyor.” Siz de şöyle dersiniz: “Çok özür dilerim; çünkü talimatlarım sözdizimi açısından doğru olsa da, ne yazık ki küçük ama fark edilmemiş bir anlam hatası içeriyordu.”

Tekrar edelim: Bu üç hata türünün hepsinde de Python, sizin tam olarak istediğiniz şeyi yapmak için elinden gelenin en iyisini yapmaya çalışıyordu.

(Ç.N.: Yukarıdaki kod bloğunda Python’a kızan kullanıcının söylediği “I hate you Python!” ve “if you come out of there, I would teach you a lesson” satırları da kodun kendisi değiştirilmediği için İngilizce kalmıştır. Türkçe karşılıkları: “Senden nefret ediyorum Python!” ve “Oradan bir çıksan sana bir ders veririm”.)

1.11 Hata Ayıklama

Python bir hata verdiğinde, hatta sizin amaçladığınızdan farklı bir sonuç ürettiğinde, hatanın nedenini bulmak için av başlar. Hata ayıklama (debugging), kodunuzdaki hatanın nedenini bulma sürecidir. Bir programda hata ayıklarken, özellikle de zorlu bir hata (bug) üzerinde çalışıyorsanız, deneyebileceğiniz dört şey vardır:

okumak (reading) Kodunuzu inceleyin, kendinize yeniden okuyun ve söylemek istediğiniz şeyi söyleyip söylemediğini kontrol edin.

çalıştırmak (running) Değişiklikler yapıp farklı sürümleri çalıştırarak deneyin. Çoğu zaman programda doğru yerde doğru şeyi ekrana yazdırırsanız sorun kendini belli eder; ama bazen yardımcı bir iskele (scaffolding) kurmak için biraz zaman harcamanız gerekir.

kafa yormak (ruminating) Düşünmeye zaman ayırın! Bu ne tür bir hata: sözdizimi mi, çalışma zamanı mı, anlam mı? Hata mesajlarından ya da programın çıktısından hangi bilgileri elde edebilirsiniz? Gördüğünüz soruna ne tür bir hata yol açmış olabilir? Sorun ortaya çıkmadan önce en son neyi değiştirdiniz?

geri çekilmek (retreating) Bir noktada yapılacak en iyi şey geri adım atmaktır: Çalışan ve anladığınız bir programa dönene kadar son değişiklikleri geri alın. Sonra yeniden inşa etmeye başlayabilirsiniz.

Programlamaya yeni başlayanlar bazen bu etkinliklerden birine takılıp kalır ve diğerlerini unuturlar. Zorlu bir hatayı bulmak için okumak, çalıştırmak, kafa yormak ve bazen de geri çekilmek gerekir. Bu etkinliklerden birinde takılıp kalırsanız, diğerlerini deneyin. Her etkinliğin kendine özgü bir tuzağı vardır.

Örneğin kodunuzu okumak, sorun bir yazım hatasıysa işe yarayabilir, ama sorun kavramsal bir yanlış anlamaysa işe yaramaz. Programınızın ne yaptığını anlamıyorsanız, onu 100 kez okuyup hatayı yine de hiç göremeyebilirsiniz, çünkü hata sizin kafanızdadır.

Deney yapmak, özellikle küçük ve basit testler çalıştırırsanız, işe yarayabilir. Ama düşünmeden ya da kodunuzu okumadan deney yaparsanız, “rastgele yürüyüş programlama” dediğim bir kalıba düşebilirsiniz; bu, program doğru şeyi yapana kadar rastgele değişiklikler yapma sürecidir. Söylemeye gerek yok, rastgele yürüyüş programlama çok uzun sürebilir.

Düşünmeye zaman ayırmalısınız. Hata ayıklama deneysel bir bilim gibidir. Sorunun ne olduğuna dair en az bir hipoteziniz olmalı. İki ya da daha fazla olasılık varsa, bunlardan birini eleyecek bir test düşünmeye çalışın.

Ara vermek düşünmeye yardımcı olur. Konuşmak da öyle. Sorunu başka birine (hatta kendinize) anlatırsanız, bazen daha soruyu sormayı bitirmeden yanıtı bulursunuz.

Ama hata sayısı çok fazlaysa ya da düzeltmeye çalıştığımız kod çok büyük ve karmaşıksa, en iyi hata ayıklama teknikleri bile işe yaramaz. Bazen en iyi seçenek geri çekilmek, yani programı çalışan ve anladığımız bir şeye ulaşana kadar sadeleştirmektir.

Programlamaya yeni başlayanlar çoğu zaman geri çekilmekte isteksizdir, çünkü (yanlış olsa bile) bir kod satırını silmeye kıyamazlar. Kendinizi daha iyi hissettirecekse, programı budamaya başlamadan önce başka bir dosyaya kopyalayın. Sonra parçaları azar azar geri yapıştırabilirsiniz.

1.12 Öğrenme yolculuğu

Kitabın geri kalanında ilerlerken, kavramlar ilk seferde birbirine pek oturmuyor gibi görünürse korkmayın. Konuşmayı öğrenirken ilk birkaç yıl yalnızca sevimli agü sesleri çıkarmanız bir sorun değildi. Basit kelimelerden basit cümlelere geçmeniz altı ay, cümlelerden paragraflara geçmeniz 5-6 yıl daha, kendi başınıza ilginç ve eksiksiz bir kısa öykü yazabilmeniz de birkaç yıl daha sürdüyse, bunda da bir sakınca yoktu.

Python’ı çok daha hızlı öğrenmenizi istiyoruz; bu yüzden sonraki birkaç bölümde her şeyi aynı anda öğretiyoruz. Ama bu, doğal gelmeye başlamadan önce sindirip anlaşılması zaman alan yeni bir dil öğrenmeye benzer. Büyük resmi oluşturan küçük parçaları tanımlarken bir yandan da size o büyük resmi göstermeye çalıştığımız için konulara tekrar tekrar döneceğiz; bu da biraz kafa karışıklığına yol açabilir. Kitap doğrusal bir sırayla yazılmış olsa da ve bir ders alıyorsanız ders de doğrusal ilerleyecek olsa da, konuları ele alış biçiminizde hiç de doğrusal olmaktan çekinmeyin. İleriye ve geriye bakın, üzerinde fazla durmadan okuyun. Daha ileri konulara, ayrıntıları tam olarak anlamadan göz gezdirerek programlamanın

“neden?”ini daha iyi kavrayabilirsiniz. Önceki konuları gözden geçirerek, hatta önceki alıştırmaları yeniden yaparak, şu anda başında durduğunuz konu biraz içinden çıkılmaz görünse bile aslında çok şey öğrendiğinizi fark edeceksiniz.

İlk programlama dilinizi öğrenirken genellikle birkaç harika “Aha!” anı yaşarsınız: Bir kayayı çekiç ve keskiyle yontmaya dalmışken başınızı kaldırır, bir adım geri çekilir ve gerçekten de güzel bir heykel yapmakta olduğunuzu görürsünüz.

Bir şey özellikle zor geliyorsa, bütün gece uyumayıp ona bakmanın genellikle bir faydası yoktur. Ara verin, biraz kestirin, bir şeyler atıştırın, sorun yaşadığınız şeyi birine (ya da belki köpeğinize) anlatın ve sonra konuya taze bir gözle geri dönün. Sizi temin ederim ki kitaptaki programlama kavramlarını öğrendiğinizde geriye dönüp baktığınızda hepsinin aslında gerçekten kolay ve zarif olduğunu, yalnızca sindirmenizin biraz zaman aldığını göreceksiniz.

1.13 Sözlükçe

ana bellek (main memory) Programları ve verileri saklar. Ana bellek, elektrik kesildiğinde içindeki bilgileri kaybeder.

anlam hatası (semantic error) Bir programın, programcının amaçladığından farklı bir şey yapmasına yol açan hata.

anlambilim (semantics) Bir programın anlamı.

ayrıştırmak (parse) Bir programı incelemek ve sözdizimsel yapısını çözümlemek.

derlemek (compile) Yüksek seviyeli bir dilde yazılmış bir programı, daha sonra yürütülmek üzere tek seferde düşük seviyeli bir dile çevirmek.

düşük seviyeli dil (low-level language) Bir bilgisayarın kolayca yürütebileceği şekilde tasarlanmış bir programlama dili; “makine kodu” ya da “assembly dili” (assembly language) olarak da adlandırılır.

etkileşimli mod (interactive mode) Python yorumlayıcısını, istemde komutlar ve ifadeler (expressions) yazarak kullanma yolu.

hata (bug) Bir programdaki yanlışlık.

ikincil bellek (secondary memory) Programları ve verileri saklar; elektrik kesildiğinde bile bilgileri korur. Genellikle ana bellekten daha yavaştır. Disk sürücüler ve USB belleklerdeki flaş bellek, ikincil bellek örnekleri arasındadır.

istem (prompt) Bir programın bir mesaj gösterip kullanıcının programa bir giriş yazmasını beklediği durum.

kaynak kod (source code) Yüksek seviyeli bir dilde yazılmış program.

makine kodu (machine code) Yazılım için en düşük seviyeli dil; merkezi işlem biriminin (CPU) doğrudan yürüttüğü dildir.

merkezi işlem birimi (central processing unit) Her bilgisayarın kalbi. Yazdığımız yazılımı çalıştıran birimdir; “CPU” ya da “işlemci” (processor) olarak da adlandırılır.

print fonksiyonu (print function) Python yorumlayıcısının ekranda bir değer göstermesini sağlayan talimat.

problem çözme (problem solving) Bir problemi formüle etme, bir çözüm bulma ve çözümü ifade etme süreci.

program Bir hesaplamayı (computation) tanımlayan talimatlar kümesi.

- taşınabilirlik (portability)** Bir programın birden fazla bilgisayar türünde çalışabilme özelliği.
- yorumlamak (interpret)** Yüksek seviyeli bir dildeki bir programı satır satır çevirerek yürütmek.
- yüksek seviyeli dil (high-level language)** Python gibi, insanların kolayca okuyup yazabileceği şekilde tasarlanmış bir programlama dili.

1.14 Alıştırmalar

Alıştırma 1: Bir bilgisayarda ikincil belleğin işlevi nedir?

- a) Programın bütün hesaplama ve mantık işlemlerini yürütmek
- b) İnternet üzerinden web sayfalarını getirmek
- c) Bilgiyi uzun süreli, elektrik kesilip yeniden gelse bile saklamak
- d) Kullanıcıdan giriş almak

Alıştırma 2: Program nedir?

Alıştırma 3: Derleyici ile yorumlayıcı arasındaki fark nedir?

Alıştırma 4: Aşağıdakilerden hangisi “makine kodu” içerir?

- a) Python yorumlayıcısı
- b) Klavye
- c) Python kaynak dosyası
- d) Bir kelime işlemci belgesi

Alıştırma 5: Aşağıdaki kodda yanlış olan nedir?

```
>>> print 'Hello world!'
File "<stdin>", line 1
    print 'Hello world!'
    ^
SyntaxError: invalid syntax
>>>
```

Alıştırma 6: Aşağıdaki Python satırının yürütülmesi bittikten sonra “x” gibi bir değişken bilgisayarın neresinde saklanır?

```
x = 123
```

- a) Merkezi işlem birimi
- b) Ana bellek
- c) İkincil bellek
- d) Giriş birimleri
- e) Çıkış birimleri

Alıştırma 7: Aşağıdaki program ekrana ne yazdırır?

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

- a) 43
- b) 42
- c) $x - 1$
- d) Hata verir, çünkü $x = x - 1$ matematiksel olarak mümkün değildir

Alıştırma 8: Aşağıdakilerin her birini bir insan yeteneğinden örnek vererek açıklayın: (1) Merkezi işlem birimi, (2) Ana bellek, (3) İkincil bellek, (4) Giriş birimi ve (5) Çıkış birimi. Örneğin, “Merkezi işlem biriminin insandaki karşılığı nedir?”