

ÖĞRENME BİRİMİ 8

HATA YAKALAMA İŞLEMLERİ

Neler Öğreneceksiniz?

Bu öğrenme birimi ile;

- Hata türlerini açıklayabilecek,
- Hangi durumlarda hata kontrolü yapmanız gerektiğini öğrenecek,
- Hata durumunda, hata yakalama ve işleme işlemlerini yapabilecek,
- Kod ile hata üretebilecek,
- Programınıza test ifadeleri yazabileceksiniz.

Anahtar Kelimeler:

Hata, yazım hatası, mantıksal hata, bug, istisnai hata, try, except, finally, raise, assert.



Hazırlık Çalışmaları

1. Kullanıcılardan alınan verilere her zaman güvenmeli miyiz?
2. Geçmişte yapılmış en büyük yazılım hatalarını araştırınız.

8. HATA YAKALAMA İŞLEMLERİ**8.1. Hata Kavramı ve Hata Türleri****8.1.1. Hata Nedir?**

Programlar, -özellikle başlangıç seviyesinde- genellikle en iyi duruma göre yazılır. Yani tüm yazılım ve donanım kaynaklarının beklenen şekilde çalışacağı ve kullanıcıların programı yazılımcının ondan beklediği şekilde kullanacağı varsayılır. Benzer şekilde bir hesaplama işleminin her durumda doğru çalışması beklenir. Ancak özellikle profesyonel seviyede, bu asla olmaması gereken bir durumdur. Programcının bütün iyi niyetli yaklaşımına rağmen işler her zaman istenildiği gibi gitmeyebilir. Bu nedenle iyi bir programcı; yazılım, donanım ve kullanıcı kaynaklı birçok hatayla karşılaşacağını bilmeli ve bunlara yönelik önlemlerini almalıdır. Programın asla kendi kontrolü dışında sonlanmasına izin vermemelidir. En kötü durumda bile kullanıcıların anlayabileceği hata mesajları vererek programı sonlandırmalıdır.

İki sayıyı toplarken herhangi bir hata olmayacağı varsayılabilir. "3+5" çoğu zaman doğru ve hatasız bir şekilde hesaplanır. Ancak farklı durumlar için her ihtimal göz önünde bulundurulmalıdır. Örneğin; herhangi bir veri hard diske kaydetmeye çalışıldığında karşılaşılabilecek durumlar şunlardır:

- Dosya adı hatalı olabilir.
- Diskte, dosyayı kaydetmeye yetecek boş yer kalmamış olabilir.
- Dosya oluşturma / yazma izni olmayabilir.
- Disk bozuk olabilir.
- İşletim sistemi doğru çalışmayabilir.
- Dosyanın kaydedilmesi anında elektrik kesintisi yaşanabilir.
- Dosya uzak bir makineye kaydedilecekse
 - o Kullanıcı adı / şifresi hatalı olabilir.
 - o Ağ bağlantısında sıkıntı olabilir.
 - o Kullanıcının yetkilendirmesinde hata olabilir.

Programcı, programını yazarken bu gibi durumları her zaman göz önünde bulundurmalıdır.

8.1.2. Hata Türleri

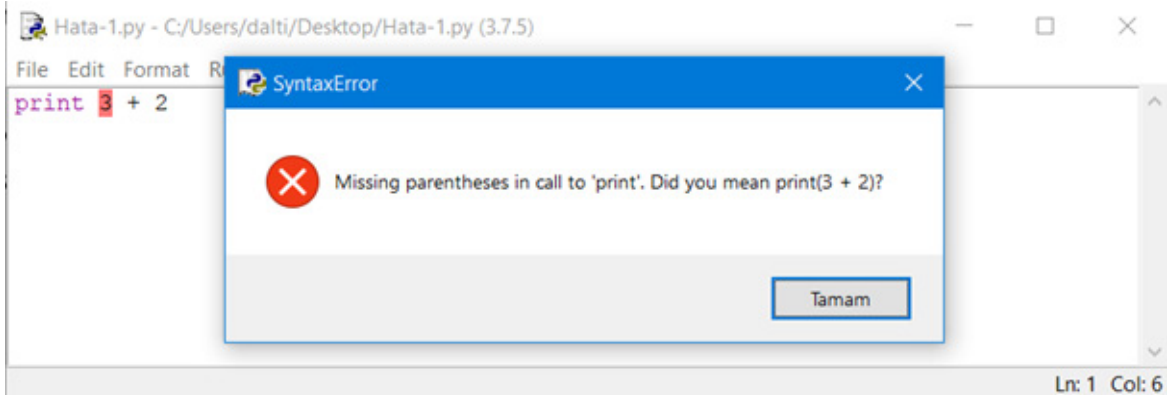
Hata kavramının sadece program dışı parametrelerde oluşabileceği düşünülmemelidir. Hataları üçe ayırmak mümkündür:

1. Programcı hataları / Yazım hataları (Syntax errors)
2. Mantıksal hatalar (Bugs)
3. İstisnai hatalar (Exceptions)

8.1.2.1. Programcı Hataları/Yazım Hataları

Programı yazan kişiden kaynaklanan hatalardır. Çoğunlukla dikkatsizlik sonucu oluşur.

Örneğin; 2 sayının toplamını hesaplayan bir program aşağıdaki şekilde yazılsın ve çalıştırılsın.



Program, doğal olarak doğru çalışmadı. Açıklama mesajında da yazdığı üzere doğrusu

```
print(3 + 2)
```

şeklinde olmalıdır. Parantez açma / kapama unutulduğu için kod doğru bir şekilde çalışmamıştır.

Bir değişkene değer atanıp ileriki satırlarda değişkenin değeri ekrana yazdırılmak istensin.

```
ogrenci_adi = "Filiz"
# Diğer kodlar
# Diğer kodlar
# Diğer kodlar
print(ogrenci_adi)
```

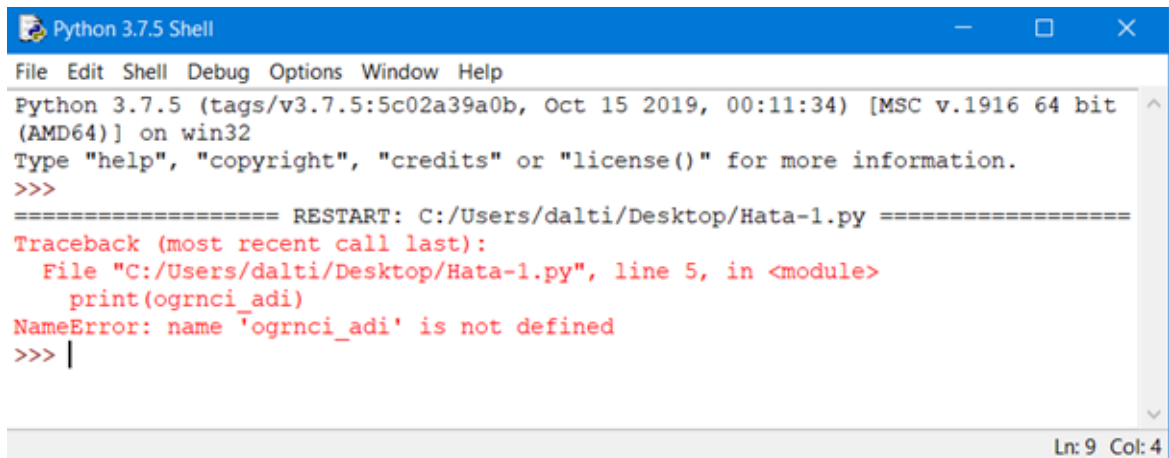
Program çalıştırıldığında yine düzgün çalışmadığı görülecektir çünkü değişkenin değeri ekrana yazdırılmak istendiğinde değişken adı programcı tarafından yanlış yazılmıştır ("ogrenci_adi" yerine "ogrn-ci_adi" yazıldığına dikkat ediniz.).

Doğrusu;

```
print(ogrenci_adi)
```

şeklinde olmalıdır.

Bu hata türü, çözülmesi en kolay hata türüdür çünkü hatalı olan satır rahatlıkla tespit edilip hata hemen düzeltilebilir.



Yukarıdaki hatalı program çalıştırıldığında, 5. satırda "ogrn-ci_adi" değişkeninin tanımlı olmadığı mesajı gösterilmektedir.

8.1.2.2. Mantıksal hatalar (Bugs)

Bu hata türünün, tespiti ve çözülmesi daha zordur. Çünkü program hata vermeden çalıştığı hâlde, programda hesaplanan sonuçlar yanlıştır. Özellikle programdaki satır sayısı arttıkça bu tür hataların tespiti de zorlaşmaktadır.

Örneğin, bir üçgenin alan hesabı formülü **"(taban kenarı * yükseklik) / 2"** şeklindedir. Bu formüle göre hesap yapan bir program yazılsın:

```
taban_kenari = 4
yukseklık = 3
alan = (taban_kenari * yukseklik) / 3
print("Alan: ", alan)
```

Programcı 3. satırda **"/2"** yazması gerekirken yanlışlıkla **"/3"** yazmıştır.

```
Python 3.7.5 Shell
File Edit Shell Debug Options Window Help
Python 3.7.5 (tags/v3.7.5:5c02a39a0b, Oct 15 2019, 00:11:34) [MSC v.1916 64 bit
(AMD64)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/dalti/Desktop/Hata-1.py =====
Alan: 4.0
>>>
```

Ln: 6 Col: 4

Her şey yolundaymış gibi görünse de üçgenin alan hesabı yanlış yapılmıştır. Program hata vermeden çalışmış ve sonucu göstermiştir. Bu hatayı fark etmek zordur ve programcı mutlaka hesaplama değerlerini test etmelidir.

Bu tür hatalara bug (böcek), ilgili hatayı bulup düzeltme işlemine de debug (ayıklama) denir.

8.1.2.3. İstisnai Hatalar

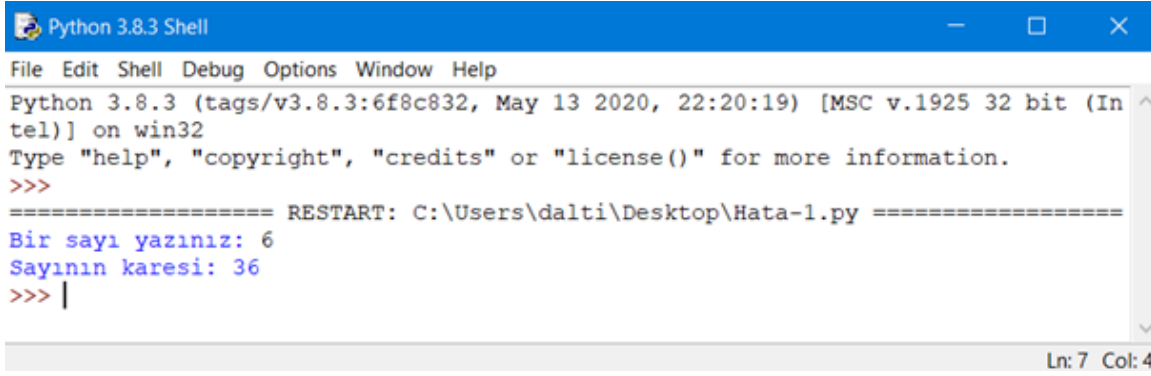
Bu tür hatalar, programın çalışması esnasında oluşan, aslında gerçekleşmesi beklenmeyen hatalardır.

Örneğin; kullanıcıdan bir sayı alıp karesini ekrana yazan bir program yazılsın.

```
sayi = int(input("Bir sayı yazınız: "))
karesi = sayi * sayi
print("Sayının karesi:", karesi)
```

Kullanıcının girdiği metinsel bilgi sayıya çevrilip sonraki satırda karesi hesaplanmakta ve son olarak ekrana yazdırılmaktadır.

Kullanıcının ilk olarak **"6"** sayısını girdiği düşünölsün.

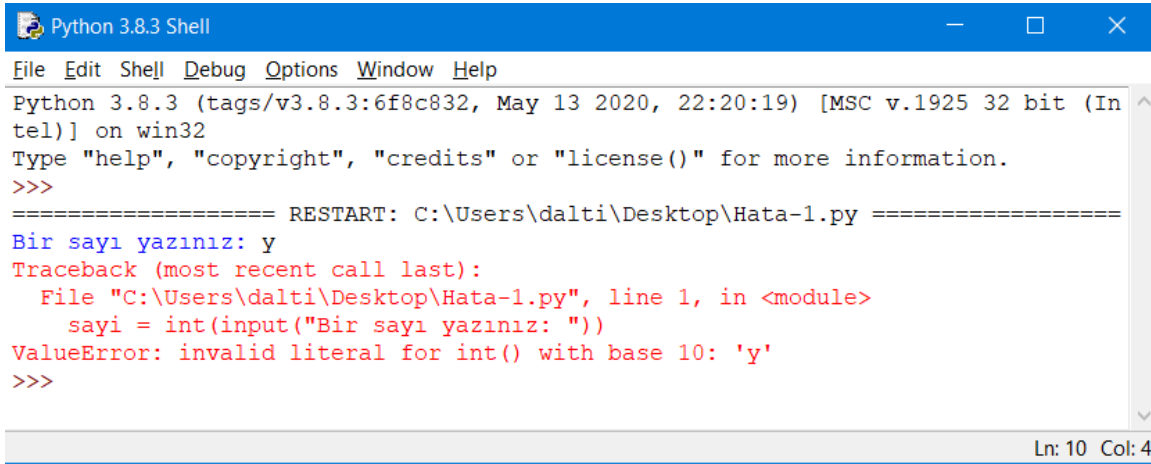


```

Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\dalti\Desktop\Hata-1.py =====
Bir sayı yazınız: 6
Sayının karesi: 36
>>> |
Ln: 7 Col: 4

```

Program düzgün ve doğru bir şekilde çalıştı. Program bir kez daha çalıştırılsın ve bu kez kullanıcının “6” yerine yanlışlıkla klavyeden “y” girdiği düşünölsün:



```

Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\dalti\Desktop\Hata-1.py =====
Bir sayı yazınız: y
Traceback (most recent call last):
  File "C:\Users\dalti\Desktop\Hata-1.py", line 1, in <module>
    sayi = int(input("Bir sayı yazınız: "))
ValueError: invalid literal for int() with base 10: 'y'
>>>
Ln: 10 Col: 4

```

Program “y” bilgisini sayıya çeviremediği için “**invalid literal for int()...**” şeklinde bir hata mesajı alınmaktadır.

Bu örnekte kullanıcının sayı girmemesiyle ortaya çıkan hata, bir istisnadır. Dolayısıyla programın istenmeden sona ermesine sebep olan bu durumun programcı tarafından kontrol altına alınması gerekir.

8.2. Hata Yakalama

Bir önceki konuda hatalardan ve hata türlerinden bahsedildi. Python programlama dilinde hata yakalama **try-except** blokları aracılığıyla yapılmaktadır.

try.. except..

En temel **try-except** bloğu şu şekildedir:

```

try:
    # hata oluşması muhtemel kod bloğu
except:
    # hata durumunda yapılacak işlemler

```

try bloğunda, hata oluşma ihtimali bulunan kodlar yazılır ve eğer bir hata oluşursa **except** bloğu devreye girer. Hata oluştuğunda **except** bloğunun devreye girmesi “**Hatayı yakalama**” olarak adlandırılır. Hata olmazsa program çalışmaya except bloğundan sonraki satırdan devam edecektir.

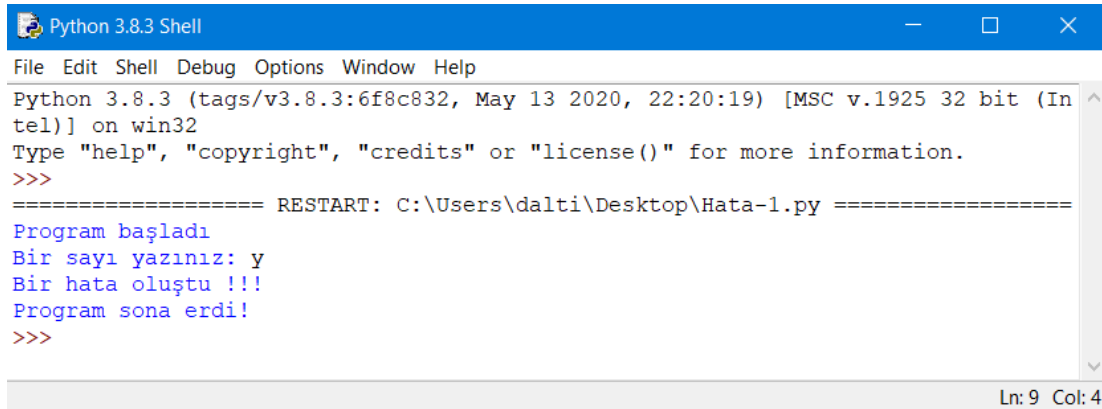
Daha önceden örnek olarak yazılan sayının kareye çevrilmesi programı, try-except bloklarıyla tekrar yazılırsa şu şekilde bir kod ortaya çıkar:

```

print("Program başladı")
try:
    sayi = int(input("Bir sayı yazınız: "))
    karesi = sayi * sayi
    print("Sayının karesi:", karesi)
except:
    print("Bir hata oluştu !!!")
print("Program sona erdi!")

```

Kullanıcının yine "6" yerine yanlışlıkla "y" girdiği düşünüldüğünde program şu şekilde çalışacaktır:



```

Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:\Users\dalti\Desktop\Hata-1.py =====
Program başladı
Bir sayı yazınız: y
Bir hata oluştu !!!
Program sona erdi!
>>>
Ln: 9 Col: 4

```

Görüldüğü üzere, program hata mesajı vererek sonlanmadı. Program, programcının kontrolü altında çalışmaya devam etti. Programcı burada, istisnai bir durum karşısında programının yarıda kesilmesini engellemiştir.

8.3. Python Hata Türleri

Python'da oluşabilecek tüm istisnai durumlar için özel hata türleri bulunmaktadır. Hata türlerinin hiyerarşik yapısı aşağıda görülebilir:

```

+-- Exception
|   +-- StopIteration
|   +-- ArithmeticError
|       |   +-- FloatingPointError
|       |   +-- OverflowError
|       |   +-- ZeroDivisionError
|   +-- AssertionError
|   +-- AttributeError
|   +-- BufferError
|   +-- EOFError
|   +-- ImportError
|   +-- LookupError
|       |   +-- IndexError
|       |   +-- KeyError
|   +-- MemoryError
|   +-- NameError
|       |   +-- UnboundLocalError

```

```

+-- OSError
|   +-- BlockingIOError
|   +-- ChildProcessError
|   +-- ConnectionError
|       +-- BrokenPipeError
|       +-- ConnectionAbortedError
|       +-- ConnectionRefusedError
|       +-- ConnectionResetError
|   +-- FileExistsError
|   +-- FileNotFoundError
|   +-- InterruptedError
|   +-- IsADirectoryError
|   +-- NotADirectoryError
|   +-- PermissionError
|   +-- ProcessLookupError
|   +-- TimeoutError
+-- ReferenceError
+-- RuntimeError
|   +-- NotImplementedError
+-- SyntaxError
|   +-- IndentationError
|   +-- TabError
+-- SystemError
+-- TypeError
+-- ValueError
|   +-- UnicodeError
|       +-- UnicodeDecodeError
|       +-- UnicodeEncodeError
|       +-- UnicodeTranslateError
+-- Warning
    +-- DeprecationWarning
    +-- PendingDeprecationWarning
    +-- RuntimeWarning
    +-- SyntaxWarning
    +-- UserWarning
    +-- FutureWarning
    +-- ImportWarning
    +-- UnicodeWarning
    +-- BytesWarning
    +-- ResourceWarning

```

8.3.1. Birden Fazla “Except” Bloğu

try-except bloğu ile tüm hataları yakalayabilirken gerektiğinde oluşabilecek hata türüne göre **except** bloğu özelleştirilebilir.

Örnek olarak 2 sayının birbirine bölümü istenildiği düşünölsün. Kullanıcıdan 2 adet sayı bilgisi istensin ve bölüm işlemi hesaplanarak ekrana yazdırılsın.

Burada hemen akla gelen 2 muhtemel durum vardır. Birincisi, kullanıcı sayı değil de başka bir şey girerse, ikincisi de kullanıcı bölen olarak “0” girerse oluşacak hatalardır. Matematikte ve bilgisayarlarda bir sayının sıfıra bölünmesi tanımsızdır. Bir sayının sıfıra bölünmesi mümkün değildir. Bu hata ihtimallerini de düşünerek yazılabilecek kod aşağıdaki şekilde olabilir:

```

try:
    sayi1 = int(input("Bölünen: "))
    sayi2 = int(input("Bölen: "))
    sonuc = sayi1 / sayi2
    print("Sonuç:", sonuc)
except ValueError:
    print("Sayı girmediniz!")
except ZeroDivisionError:
    print("Sayıyı 0'a bölemezsiniz!")

```

Programa farklı değerler girerek birkaç kez çalıştırıldığında aşağıdaki gibi bir sonuç elde edilir:

```

Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/dalti/Desktop/Bolum.py =====
Bölünen: 10
Bölen: 2
Sonuç: 5.0
>>>
===== RESTART: C:/Users/dalti/Desktop/Bolum.py =====
Bölünen: asd
Sayı girmediniz!
>>>
===== RESTART: C:/Users/dalti/Desktop/Bolum.py =====
Bölünen: 10
Bölen: asd
Sayı girmediniz!
>>>
===== RESTART: C:/Users/dalti/Desktop/Bolum.py =====
Bölünen: 10
Bölen: 0
Sayıyı 0'a bölemezsiniz!
>>> |
Ln: 22 Col: 4

```

Sonuç ekranında da görüldüğü üzere hem sayıya çevrilme hatası hem de sıfıra bölme hatası ayrı ayrı yakalanmıştır. Bu örnekten yola çıkarak farklı türde hatalar oluştuğunda farklı işlemler yaptırılması mümkündür denilebilir. Genel yapı şu şekildedir:

```

try:
    # çalıştırılacak kodlar
except hatatipi1:
    # Yapılacak işlemler
except hatatipi2:
    # Yapılacak işlemler
except hatatipi3:
    # Yapılacak işlemler

```

Eğer birden fazla hatada aynı işlemlerin yapılması isteniyorsa şu şekilde de bir kullanım mümkündür:


```
try:
    # çalıştırılacak kodlar
except (hatatipi1, hatatipi2):
    # Yapılacak işlemler
except hatatipi3:
    # Yapılacak işlemler
```

8.3.2. “as” İfadesi ile Orijinal Hata Mesajı Gösterme

except bloğunda istenilen hata mesajı gösterilebildiği gibi Python tarafından oluşturulan orijinal hata mesajı da gösterilebilir.

Bunun için **as** deyimini kullanılır.

```
try:
    sayi1 = int(input("1. sayı: "))
    sayi2 = int(input("2. sayı: "))
    toplam = sayi1 + sayi2
    print("Toplam:", toplam)
except ValueError as hata:
    print("Sayı girmediniz!")
    print("Orijinal hata mesajı:", hata)
```

except bloğunda, oluşan hata bilgisi “hata” değişkeninde tutulmaktadır.

8.3.3. “finally” Bloğu

try bloğunda yazılan kodlarda hata olsa da olmasa da çalışması istenilen kodlar **finally** bloğuna yazılır.

Genel yapısı şu şekildedir:

```
try:
    # çalıştırılacak kodlar
except:
    # hata oluştuğunda yapılacak işlemler
finally:
    # hata olsa da olmasa da çalışacak kodlar
```

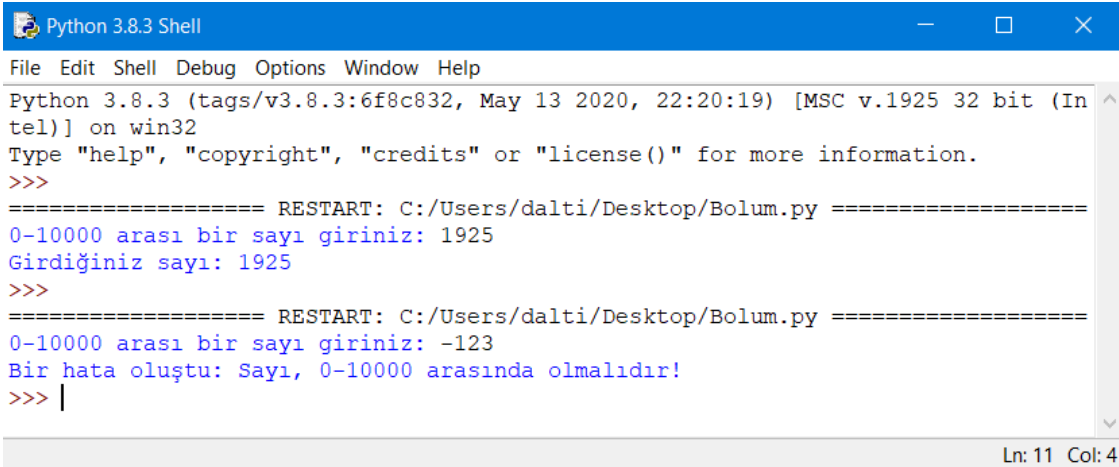
Özellikle dosya işlemlerinde, veri tabanı işlemlerinde kullanılması gereken bir yapıdır. Bir dosya ya da veri tabanı bağlantısı açıldığında mutlaka bir şekilde kapatılmalıdır.

8.3.4. “raise” İfadesi

Yazılan kodlarda kasıtlı olarak bir hata oluşturulması istenebilir. Örneğin; kullanıcıdan 0-100 arası bir sayı girmesi istendiğinde, kullanıcı “-5” ya da “110” değerini girerse, Python açısından hiçbir hata olmamasına rağmen, istenilen değer aralığında olmadığı için bir hata üretilebilir. (Bu durum bazı kaynaklarda “**hata fırlatma**” olarak da geçmektedir.) Oluşan bu hatayı da **except** bloğu içinde yakalamak mümkündür.

```
try:
    sayi = int(input("0-10000 arası bir sayı giriniz: "))
    if(sayi not in range(0, 10001)):
        raise Exception("Sayı, 0-10000 arasında olmalıdır!")
    print("Girdiğiniz sayı:", sayi)
except Exception as hata:
    print("Bir hata oluştu:", hata)
```

raise Exception("...") satırında, istenilen hata mesajı üretilmiş ve **except** bloğunda yakalanmıştır.



```
Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/dalti/Desktop/Bolum.py =====
0-10000 arası bir sayı giriniz: 1925
Girdiğiniz sayı: 1925
>>>
===== RESTART: C:/Users/dalti/Desktop/Bolum.py =====
0-10000 arası bir sayı giriniz: -123
Bir hata oluştu: Sayı, 0-10000 arasında olmalıdır!
>>> |
```

Ln: 11 Col: 4

8.3.5. "assert" İfadesi

Program yazılırken programın herhangi bir satırında bir değişkenin istenilen değere sahip olup olmadığının test edilmesi gerekebilir. Bunun için **print** komutu ile değişkenin değerini ekrana yazdırmak bir çözüm olsa da çok fazla ekran çıktısı arasında istediğimiz değeri görmek zorlaşacağı için sadece test ifadesi sağlanmadığında bir hata üretmek **assert** ifadesi ile mümkündür.

Genel olarak kullanımı şu şekildedir:

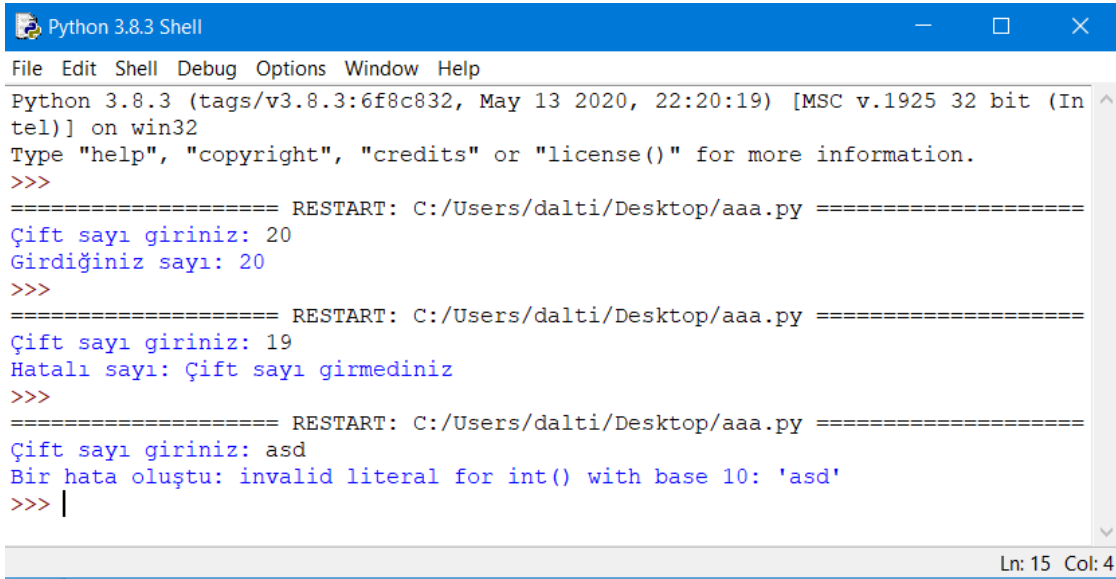
```
assert koşul, mesaj
```

Koşul ifadesi **true** ya da **false** değerlerinden birini üreten bir ifadedir ve eğer bu değer **true** ise bir sonraki satırdan program devam edecek; eğer **false** ise **"mesaj"** bilgisi kullanılarak **"AssertionError"** tipinde bir hata oluşturulacaktır.

Örneğin kullanıcıdan bir çift sayı girmesi istensin:

```
try:
    cift_sayi = int(input("Çift sayı giriniz: "))
    assert cift_sayi % 2 == 0, "Çift sayı girmediniz"
    print("Girdiğiniz sayı:", cift_sayi)
except AssertionError as hata:
    print("Hatalı sayı:", hata)
except Exception as hata:
    print("Bir hata oluştu:", hata)
```

"`cift_sayi % 2 == 0`" ifadesi ile kullanıcının girdiği değerin çift sayı olup olmadığı test edilmektedir. Eğer çift sayı ise bir sonraki satırda ekrana yazdırılacak, çift sayı değilse de **assert** ifadesi otomatik olarak "**AssertionError**" tipinde bir hata oluşturacaktır. **Except** bloğu içinde **as** ifadesi ile oluşturulan hata mesajının ekrana yazdırıldığı görülebilir.



```
Python 3.8.3 Shell
File Edit Shell Debug Options Window Help
Python 3.8.3 (tags/v3.8.3:6f8c832, May 13 2020, 22:20:19) [MSC v.1925 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license()" for more information.
>>>
===== RESTART: C:/Users/dalti/Desktop/aaa.py =====
Çift sayı giriniz: 20
Girdiğiniz sayı: 20
>>>
===== RESTART: C:/Users/dalti/Desktop/aaa.py =====
Çift sayı giriniz: 19
Hatalı sayı: Çift sayı girmediniz
>>>
===== RESTART: C:/Users/dalti/Desktop/aaa.py =====
Çift sayı giriniz: asd
Bir hata oluştu: invalid literal for int() with base 10: 'asd'
>>> |
```

Ln: 15 Col: 4

ÖLÇME VE DEĞERLENDİRME 8

1. Bir try-except bloğunda kaç tane except bloğu olabilir?
2. Aşağıdaki kod bloğu geçerli midir?

```
try:
    # Kodlar...
except:
    # Kodlar...
finally:
    # Kodlar...
```

3. Bir except bloğunda birden fazla hata yakalanabilir mi? Örnek vererek anlatınız.
4. Aşağıdaki kod bloğunun çıktısı ne olur?

```
def func():
    try:
        return 1
    finally:
        return 2
k = func ()
print(k)
```

- A) 0 B) 1 C) 2 D) 3 E) Hata verir.

5. 1 ifadesi ne sonuç verir?

- A) True B) False C) Hiçbir sonuç vermez.
D) TypeError hatası E) ValueError hatası

6. Aşağıdaki kodun çıktısı ne olur? Satır satır açıklayınız.

```
print("Program başladı")
try:
    raise Exception('Bir hata oluştu!')
except:
    print("Except bloğuna geldik.")
    raise
finally:
    print("Finally bloğuna geldik.")
print("Program bitti")
```

NOT: Cevaplarınızı cevap anahtarıyla karşılaştırınız. Yanlış cevap verdiğiniz ya da cevap verirken tereddüt ettiğiniz sorularla ilgili konuları veya faaliyetleri geri dönerek tekrarlayınız. Cevaplarınızın tümü doğru ise bir sonraki öğrenme birimine geçiniz.