PYTHON PROGRAMLAMA DİLİ

HATA YAKALAMA
try-except-else-finally

Dr. Öğr. Üyesi M. Ozan AKI

HATA YAKALAMA

Kodumuzu yazarken yazım hatası yaptığımızda ve çalışma sırası o satıra geldiğinde Python hata verecektir. Bu tür hatalara yazım hataları (**Syntax Error**) adı verilir.

Ancak yazdığımız kodda hiç bir yazım hatası olmamasına rağmen çalışma sırasında farklı nedenlerle de program hata ile karşılaşabilir.

Örneğin;

Kullanıcıdan sayı girmesini beklerken harfler girebilir.

Kullanıcı, girmesi gereken bir veriyi boş geçebilir.

İnternetten ulaşmaya çalıştığımız bir veri, sunucu cevap vermediği için gelmeyebilir.

Network üzerinden veri gönderip alırken network kablosu çıkarılabilir.

Bir dosya kaydetmek istediğimizde, diskte yeterli alan kalmamış olabilir.

Bir dosya okumak istediğimizde, işletim sistemi yetkisiz erişimden dolayı dosyayı okumayı engelleyebilir.

HATA YAKALAMA

Buna göre hataları çeşitli kategorilere ayırabiliriz;

1) Yazım Hataları (**Syntax Error**)
2) Mantık (Algoritma) Hataları
3) İstisnai Hatalar (**Exceptions**)

Bu hata türlerinden Yazım hataları (**Syntax Error**) ve Mantık hataları, kodu tasarlayıp yazan programcıdan kaynaklanan hatalardır.

İstisnai hatalar ise, öngörülmesi gereken ancak öngörülmeden her şeyin yolunda gideceğini varsayan, kodu tasarlayıp kodlayan programcının öngörü eksiklikleridir.

HATA YAKALAMA

Python istisnai (**Exception**) hatalarının tüm listesi

http://docs.python.org/3/library/exceptions.html

Bağlantısından görülebilir.

```
BaseException
 ├── BaseExceptionGroup
 ├── GeneratorExit
 ├── KeyboardInterrupt
 ├── SystemExit
 └── Exception
      ├── ArithmeticError
      │    ├── FloatingPointError
      │    ├── OverflowError
      │    └── ZeroDivisionError
      ├── AssertionError
      ├── AttributeError
      ├── BufferError
      ├── EOFError
      ├── ExceptionGroup [BaseExceptionGroup]
      ├── ImportError
      │    └── ModuleNotFoundError
      ├── LookupError
      │    ├── IndexError
      │    └── KeyError
      ├── MemoryError
      ├── NameError
      │    └── UnboundLocalError
```

HATA YAKALAMA – try-except

Yazım ve mantık hataları dışında, çalışma sırasında ortaya çıkan hataları yakalamak ve program akışını kesmeden bu hatalarla ilgili gereğini yapmayı sağlayan yapı;

try-except

Yapısı olup, **try** ve **except** ifadeleri arasında oluşabilecek herhangi hatayı yakalayıp bununla ilgili özel bir kod bloğunun çalıştırılmasını sağlar.

```
try:
    kodlar
    kodlar
    ...
except:
    hata olduğunda çalışacak kodlar
```

HATA YAKALAMA – try-except

Eğer özel bir hata türünü yakalamak ve bununla ilgili yapmak istersek, **except** ifadesinin yanına beklediğimiz hata türünü yazabiliriz.

```
try:
    kodlar
    kodlar
    ...
except ZeroDivisonError:
    sıfıra bölüm hatası olduğunda
    çalışacak kodlar
```

HATA YAKALAMA – try-except-as

Eğer yakaladığımız hata ile ilgili hata mesajı, hatanın kaynağı gibi detaylara ulaşmak istersek **except** ifadesinin devamına;

as takma_ad

ifadesi eklenerek, hata ile ilgili detayların takma ad ile verilen değişkene aktarılmasını sağlarız.

Takma ad yerine genel e ismi kullanılır.

Hata ile ilgili detaylı bilgi **e** nesnesi içinde döndürülür.

```
try:
    kodlar
    kodlar
    ...
except ZeroDivisonError as e:
    sıfıra bölüm hatası olduğunda
    çalışacak kodlar
```

HATA YAKALAMA – try-except

Eğer birkaç farklı hata türü yakalamak ve bununla ilgili yapmak istersek, **except** ifadesinin yanına beklediğimiz hata türlerini aralarına virgül koyarak yazabiliriz.

```
try:
    kodlar
    kodlar
    ...
except ZeroDivisonError:
    sıfıra bölüm hatası olduğunda çalışacak kodlar
except ValueError:
    değer hatası olduğunda çalışacak kodlar
```

HATA YAKALAMA – try-except

Eğer birkaç farklı hata türü yakalamak ve bununla ilgili yapmak istersek, **except** ifadesinin yanına beklediğimiz hata türlerini bir **Tuple** olarak aralarına virgül koyarak belirtebiliriz.

Gruplandırdığımız hatalardan herhangi biri oluştuğunda bu **except** ifadesi ile hatalar yakalanır.

```
try:
    kodlar
    kodlar
    ...
except (ZeroDivisonError, ValueError):
    sıfıra bölüm ya da değer hatası
    olduğunda çalışacak kodlar
```

HATA YAKALAMA – try-except-as

Tuple olarak verdiğimiz hata türleri ile ilgili detaylı bilgi almak istersek, **except** ifadesinin devamına

as takma_ad

İfadesi ekleyerek hata ile ilgili detaylı bilgi elde edebiliriz.

Hata ile ilgili detaylar, belirttiğimiz isim altında erişilebilir durumda olacaktır.

```
try:
    kodlar
    kodlar
    ...
except (ZeroDivisonError, ValueError) as e:
    sıfıra bölüm ya da değer hatası
    olduğunda çalışacak kodlar
```

HATA YAKALAMA – else

Eğer hiç bir hata oluşmadığında belirli bir kod bloğu çalıştırmak istersek, **except** ifadesinden sonra bir

else:

İfadesi kullanarak her şeyin yolunda gitmesi durumunda çalıştırılacak kodları bu blok altına yazarız.

```
try:
    kodlar
    kodlar
    ...
except:
    hata olduğunda çalışacak kodlar
else:
    hiç hata olmazsa çalışacak kodlar
```

HATA YAKALAMA - finally

Eğer herhangi hata olsun yada hiç hata oluşmasın, her halükarda çalıştırılmasını istediğimiz kodlar varsa, bu kodları, try bloğu devamında yazacağımız

finally:

İfadesi kullanarak tanımlayabiliriz. Bu blok altına yazdığımız kodlar, hata olsun ya da olmasın, try bloğu sonlanmadan önce çalıştırılır.

Burada, try bloğu içinde başlatılan bazı işlemlerin sonlandırılması amacıyla kullanılır.

```
try:
    kodlar
    kodlar
    ...
except:
    hata olduğunda çalışacak kodlar
else:
    hiç hata olmazsa çalışacak kodlar
finally:
    bu kodlar her zaman calisir
```

HATA YAKALAMA - raise

Bazen kontrol edebildiğimiz bir hata oluştuğunda kendi istisnai durumumuzu oluşturmak isteyebiliriz.

Bunun için **try…except** bloğu içindeki kodumuzda

raise

İfadesi ile bir istisna (**exception**) çıkarabiliriz ve kod hemen **except**: bloğuna atlar.

```
try:
    kodlar
    if hata_test:
        raise Exception("Hata oluştu")
    kodlar
    ...
except:
    hata olduğunda çalışacak kodlar
```

HATA YAKALAMA - assert

raise ifadesine benzer şekilde, belirli bir koşula bağlı olarak koşul yanlış olduğunda hata çıkaran

assert

İfadesi de genelde programlardaki olası hataları tespit etmek için kullanılır.

Eğer bir değişkende uygun olmayan ya da beklenmeyen bir veri geldiğinde program düzgün çalışamayacak ise, bu durumda değişkenin değeri assert ile kontrol edilerek uygun olmayan değerlerde hata çıkarması sağlanır.

```
try:
    kodlar
    assert sayi%2!=0,"sayi tek degil"
    kodlar
    ...
except AssertionError:
    assert olduğunda çalışacak kodlar
except:
    hata olduğunda çalışacak kodlar
```