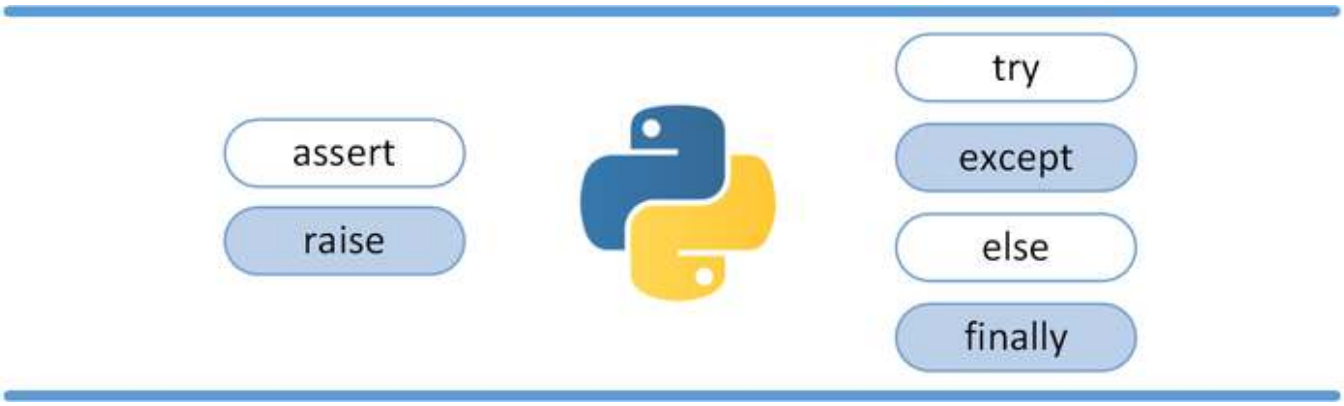
Python Exceptions

یک برنامه پایتون به محض برخورد با خطا متوقف می شود. در پایتون، خطا می تواند یک خطای نحوی یا یک استثناء باشد. در این مقاله، شما خواهید دید که استثناء چیست و چگونه از خطای نحوی متفاوت است. پس از آن، در مورد برانگیختن استثناء ها و ادعاها صحبت خواهید کرد. سپس، با نمایش بلوک try و except به پایان خواهید رسید.



Exception ها در مقابل خطای syntax

خطاهای syntax هنگامی رخ می دهند که پارسر یک بیانیه نادرست را تشخیص می دهد. به عنوان مثال زیر را مشاهده کنید:

In [23]:

```
1 print(0 / 0 ))
```

```
  Cell In[23], line 1
    print(0 / 0 ))
                  ^
SyntaxError: unmatched ')'
```

نشانگر نشان می دهد کجا پارسر به خطای نحوی برخورد کرده است. در این مثال، یک پرانتز بیش از حد وجود دارد. آن را حذف کنید و کد خود را دوباره اجرا کنید:

In [24]:

```
1 print(0 / 0)
```

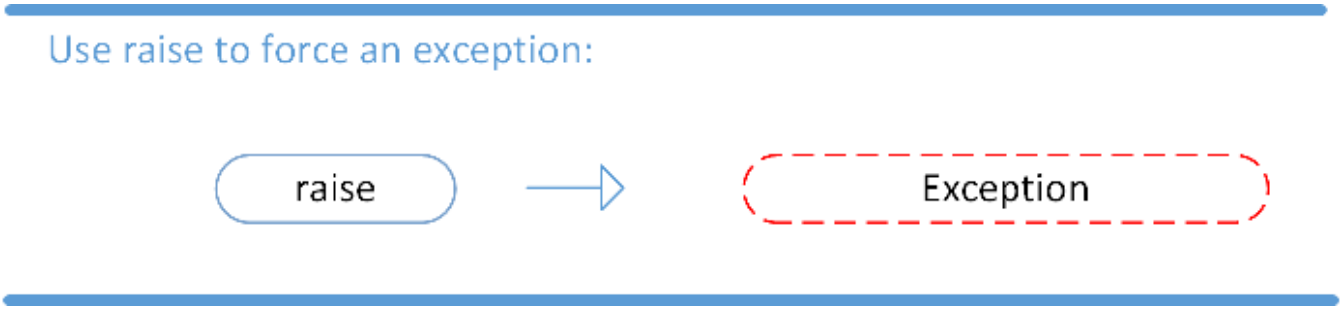
```
---------------------------------------------------------------------------
-
ZeroDivisionError                         Traceback (most recent call las
t)
Cell In[24], line 1
----> 1 print(0 / 0)

ZeroDivisionError: division by zero
```

این بار، شما با خطای استثناء مواجه شدید. این نوع خطا هرگاه کد پایتون نحوی صحیح باعث بروز خطا شود رخ می دهد. آخرین خط پیام نشان می دهد چه نوع خطای استثناء با آن روبرو شده اید.

بجای نمایش پیام خطای استثناء، پایتون جزئیات نوع خطای استثناء را که با آن روبرو شده است، توضیح می دهد. در این مورد، ZeroDivisionError بود. پایتون با استثناء های داخلی مختلف همراه است و همچنین امکان ایجاد استثناء های تعریف شده توسط کاربر وجود دارد.

ایجاد یک Exception

می توانیم از raise برای پرتاب یک استثناء در صورت بروز یک شرط استفاده کنیم. این عبارت می تواند با یک استثناء سفارشی تکمیل شود.



اگر می خواهید هنگام بروز یک شرط خاص با استفاده از raise خطا را پرتاب کنید، می توانید به این صورت عمل کنید:

In [25]:

```
1 x = 10
2
3 if x > 5:
4     raise Exception(f'X should not exceed 5. The value of x was: {x}')
```

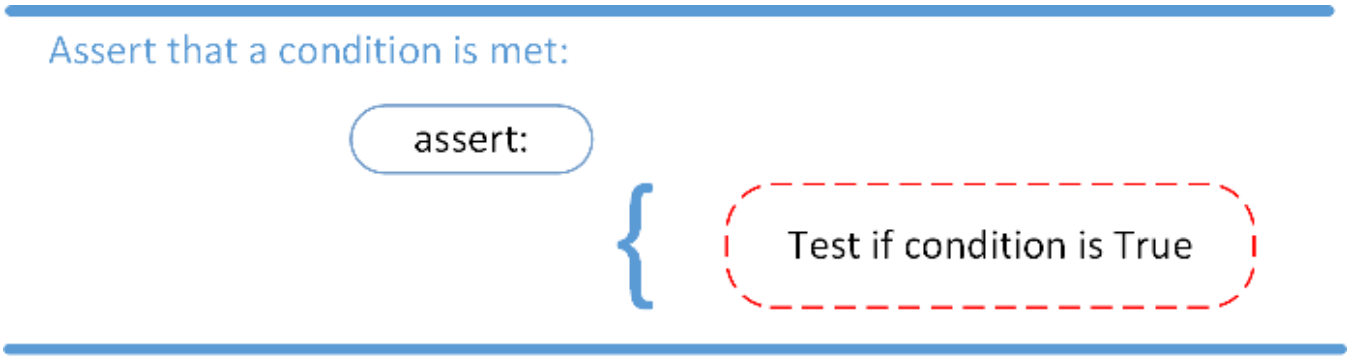
```
---------------------------------------------------------------------------
-
Exception                                 Traceback (most recent call las
t)
Cell In[25], line 4
      1 x = 10
      3 if x > 5:
----> 4     raise Exception(f'X should not exceed 5. The value of x was:
{x}')

Exception: X should not exceed 5. The value of x was: 10
```

وقتی این کد را اجرا می کنید، خروجی به شکل زیر خواهد بود:

برنامه به پایان می رسد و استثناء ما را به صفحه نمایش می کشاند و درباره آنچه اشتباه رفت، راهنمایی هایی ارائه می دهد.

AssertionError Exception

به جای اینکه منتظر بمانید تا برنامه در نیمه راه خراب شود، می توانید با یک ادعا کار را در پایتون شروع کنید. ما ادعا می کنیم که یک شرط خاص برآورده شده است. اگر این شرط به درستی ثابت شود، آنگاه عالی است! برنامه می تواند ادامه پیدا کند. اگر شرط ثابت شود که نادرست است، می توانید برنامه را به یک استثناء AssertionError پرتاب کنید.



به مثال زیر نگاه کنید، جایی که ادعا شده است کد بر روی یک سیستم لینوکس اجرا خواهد شد:

In [26]:

```
1 import sys
2
3 assert ('linux' in sys.platform), "This code runs on Linux only."
```

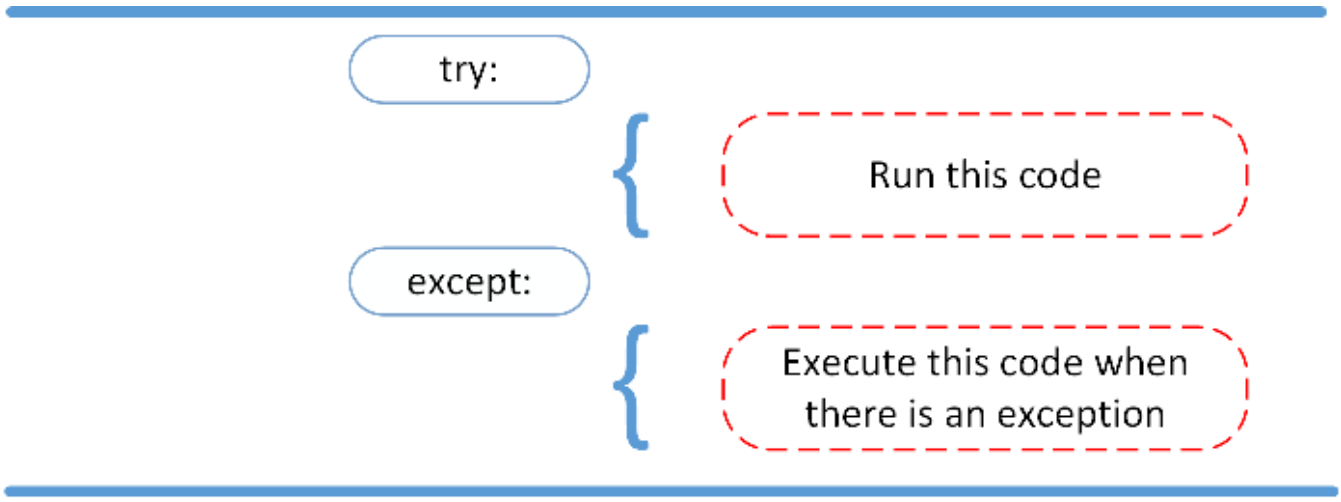
```
---------------------------------------------------------------------------
AssertionError                            Traceback (most recent call las
t)
Cell In[26], line 3
      1 import sys
----> 3 assert ('linux' in sys.platform), "This code runs on Linux only."

AssertionError: This code runs on Linux only.
```

اگر این کد را روی یک سیستم لینوکس اجرا کنید، ادعا عبور می کند. اگر شما این کد را روی یک سیستم ویندوز اجرا کنید، نتیجه ادعا False خواهد بود و نتیجه به شرح زیر خواهد بود.

در این مثال، پرتاب یک استثناء AssertionError آخرین چیزی است که برنامه انجام خواهد داد. برنامه به سرعت متوقف می شود و ادامه نمی دهد. اگر همین چیزی نباشد که می خواهید چه؟

مدیریت Exception ها از طریق بلوک های try & except

بلوک try و except در پایتون برای گرفتن و کنترل استثناء ها استفاده می شود. پایتون کد را که پس از عبارت try قرار دارد به عنوان بخشی "عادی" از برنامه اجرا می کند. کدی که پس از عبارت except قرار دارد، پاسخ برنامه به هر استثناء در بخش try قبلی است.



همانطور که قبلاً دیدید، هنگامی که کد نحوی به خطا برخورد می کند، پایتون یک خطای استثناء را پرتاب می کند. این خطای استثناء در صورت عدم برخورد با آن، برنامه را خراب می کند. شرط except نحوه پاسخ برنامه شما به استثناء ها را

In [27]:

```
def linux_interaction():
    assert ('linux' in sys.platform), "This code runs on Linux only."
    print('Doing Something')
```

تابع linux_interaction() فقط در سیستم عامل لینوکس قابل اجرا است. اگر شما این تابع را در یک سیستم عامل غیر از لینوکس فراخوانی کنید، assert در این تابع یک استثناء AssertionError را پرتاب می کند.

شما می توانید با استفاده از کد زیر تابع را امتحان کنید:

In [28]:

```
try:
    linux_interaction()
except:
    pass
```

روشی که شما در اینجا خطای خود را دست کاری کرده اید، به صورت دادن یک pass است. اگر شما این کد را در یک ماشین ویندوز اجرا کنید، خروجی خالی دریافت خواهید کرد.

شما هیچ چیز نگرفتید. نکته مثبت در اینجا این است که برنامه خراب نشده است. اما خوب بود مشخص شود هرگز هنگام اجرای کد شما، چه نوع استثناءی رخ داده است. به این منظور، می توانید pass را به چیزی تغییر دهید که پیام اطلاعاتی تولید کند، مانند زیر:

In [30]:

```
try:
    linux_interaction()
except:
    print('Linux function was not executed')
```

```
Linux function was not executed
```

هنگامی که یک استثناء در یک برنامه در حال اجرا این تابع رخ می دهد، برنامه شما را در مورد اینکه فراخوانی تابع موفق نبود، مطلع می کند.

آنچه که شما نتوانستید ببینید، نوع خطایی بود که به دلیل فراخوانی تابع پرتاب شده بود. برای دیدن دقیق آنچه که اشتباه رفت، شما باید خطایی را که تابع پرتاب کرده است، گرفتار کنید.

کد زیر یک مثال است که در آن شما AssertionError را ضبط و پیام آن را به صفحه نمایش خروجی می دهید:

In [31]:

```
try:
    linux_interaction()
except AssertionError as error:
    print(error)
    print('Linux function was not executed')
```

```
This code runs on Linux only.
Linux function was not executed
```

پیام اول AssertionError است که شما را مطلع می کند تابع فقط در یک ماشین لینوکس قابل اجرا است. پیام دوم به شما می گوید کدام تابع اجرا نشده است.

در مثال قبل، شما یک تابع را که خودتان نوشته بودید فراخوانی کردید. هنگامی که تابع را اجرا کردید، استثناء AssertionError را گرفتید و آن را به صفحه نمایش چاپ کردید.

اینجا مثال دیگری وجود دارد که در آن یک فایل را باز می کنید و از استثناء داخلی استفاده می کنید:

In [32]:

```
try:
    with open('file.log') as file:
        read_data = file.read()
except:
    print('Could not open file.log')
```

```
Could not open file.log
```

این یک پیام اطلاعاتی است و برنامه ما همچنان ادامه خواهد داد. در مستندات پایتون https://docs.python.org/3/library/exceptions.html، (https://docs.python.org/3/library/exceptions.html%D8%8C) می توانید ببینید که تعداد زیادی استثناء داخلی وجود دارد که می توانید در اینجا استفاده کنید. یک استثناء که در آن صفحه توصیف شده است، به شرح زیر است:

```
Exception FileNotFoundError
```

وقتی درخواست فایل یا پوشه ای که وجود ندارد، ارسال می شود، این استثناء برای شما پرتاب می شود. این استثناء با errno ENOENT مطابقت دارد.

برای گرفتن این نوع استثناء و چاپ آن روی صفحه، می توانید از کد زیر استفاده کنید:

In [36]:

```
try:
    with open('file.log') as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
    print('Could not open file.log')

print(5 ** 5)
```

```
[Errno 2] No such file or directory: 'file.log'
Could not open file.log
3125
```

"شما می‌توانید بیش از یک فراخوانی تابع در شرط try خود داشته باشید و پیش بینی کنید که چندین استثناء را می‌گیرید. چیزی که در اینجا باید توجه داشت این است که کد در شرط try به محض بروز استثناء متوقف می‌شود."

توجه: پرتاب استثناء Exception تمام خطاها را مخفی می کند ... حتی آن هایی که کاملاً غیرمنتظره هستند. به همین دلیل باید از بلوک های except بدون پارامتر در برنامه های پایتون خودداری کنید. به جای آن، شما می خواهید به کلاس های استثناء خاصی که می خواهید آن ها را پرتاب و پردازش کنید، ارجاع دهید. شما می توانید در این آموزش بیشتر درباره اینکه چرا این یک ایده خوب است، بیاموزید.

به کد زیر نگاه کنید. در اینجا، شما ابتدا تابع linux_interaction() را فراخوانی می کنید و سپس سعی می کنید یک فایل باز کنید:

In [38]:

```
try:
    linux_interaction()
    with open('file.log') as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
except AssertionError as error:
    print(error)
    print('Linux linux_interaction() function was not executed')
```

```
[Errno 2] No such file or directory: 'file.log'
```

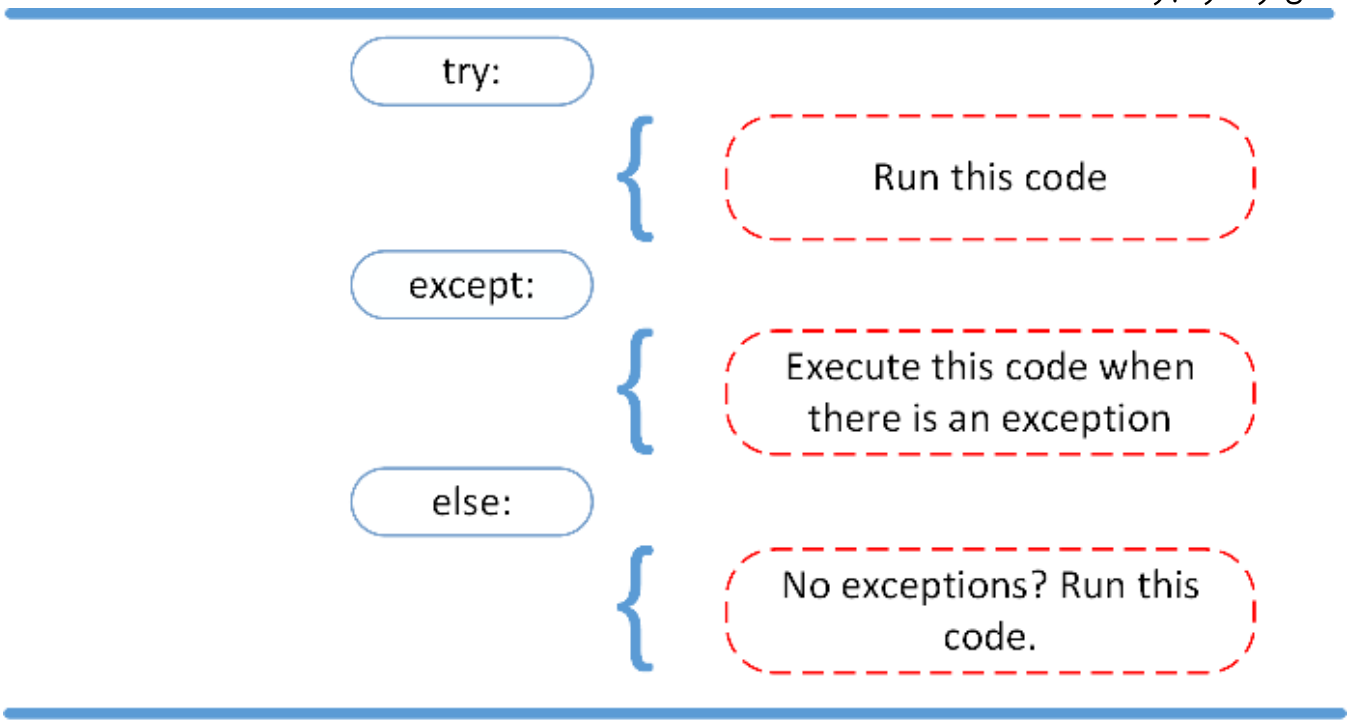
در داخل بلوک try با اولین exception اجرای دستورات متوقف خواهد شد.

نکات کلیدی که به خاطر داشتن آنها مفید است:

- یک بلوک try تا زمانی که اولین استثنا رخ دهد، اجرا می‌شود.
- در داخل بخش except یا exception handler، شما مشخص می‌کنید که برنامه به استثنا چگونه پاسخ می‌دهد.
- شما می‌توانید چندین استثنا را پیش بینی کرده و تفاوت رفتار برنامه در برابر آن‌ها را تعیین کنید.
- بهتر است از bare except clauses خودداری کنید.

عبارت else

در پایتون، با استفاده از دستور else، می‌توانید به برنامه دستوری را بدهید تا فقط در صورت عدم وجود استثنا، یک بلوک خاص از کد را اجرا کند.



به مثال زیر نگاه کنید:

In [39]:

```
def win32_interaction():
    assert ('win32' in sys.platform), "This code runs on Windows only."
    print('Doing Something')
```

In [40]:

```
try:
    win32_interaction()
except AssertionError as error:
    print(error)
else:
    print('Executing the else Clause.')
```

```
Doing Something
Executing the else Clause.
```

بلوک else اجرا شد، زیرا برنامه با هیچ استثنایی مواجه نشد.

همچنین می‌توانید کد را درون بلوک else اجرا کرده و در آنجا نیز استثناهای ممکن را گرفته و پردازش کنید:

In [41]:

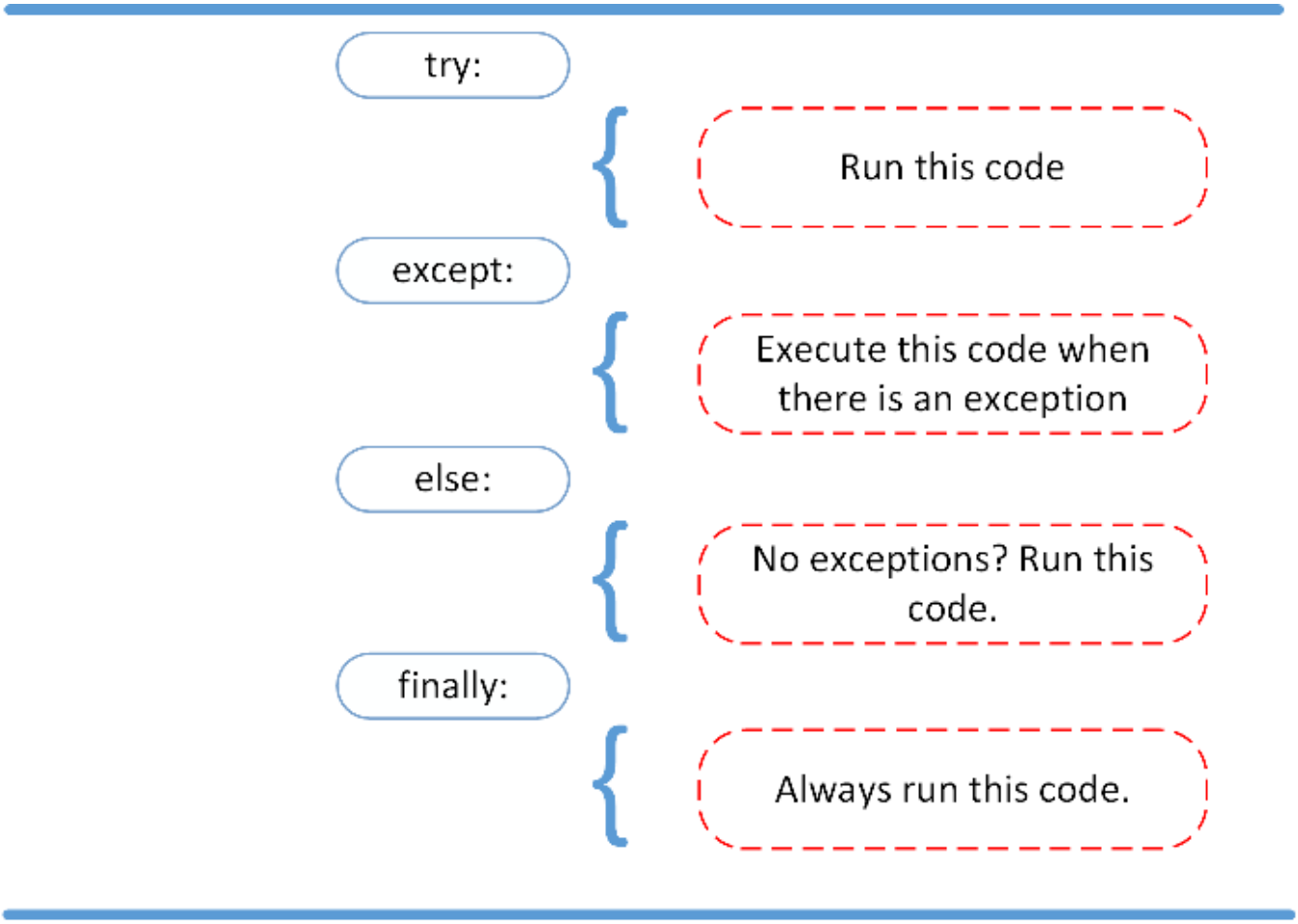
```
try:
    win32_interaction()
except AssertionError as error:
    print(error)
else:
    try:
        with open('file.log') as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
```

```
Doing Something
[Errno 2] No such file or directory: 'file.log'
```

از خروجی می‌توانید ببینید که تابع win32_interaction() اجرا شده است. زیرا هیچ استثنایی وجود نداشت، تلاش برای باز کردن فایل file.log انجام شد. این فایل وجود نداشت و به جای باز کردن فایل، استثنای FileNotFoundError را گرفتید.

عبارت finally

تصور کنید همیشه باید عملیاتی را برای جمع بندی نهایی، پس از اجرای کد خود پیاده کنید. با استفاده از بخش finally در پایتون می‌توانید این کار را انجام دهید.



به مثال زیر نگاهی بیندازید:

In [43]:

```
try:
    win32_interaction()
except AssertionError as error:
    print(error)
else:
    try:
        with open('file.log') as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
finally:
    print('Cleaning up, irrespective of any exceptions.')
```

```
Doing Something
[Errno 2] No such file or directory: 'file.log'
Cleaning up, irrespective of any exceptions.
```

مثال

ما می توانیم این را به همراه except استفاده کنیم. بیایید یک مثال جدید را ببینیم که با ورود اشتباه کاربر مواجه می شود:

In [44]:

```
def askint():
    try:
        val = int(input('Please enter an integer: '))
    except:
        print('Looks like you did not enter an integer!')
    finally:
        print('Finally, I executed!!')

    print(val)
```

In [45]:

```
askint()
```

```
Please enter an integer: 5
Finally, I executed!!
5
```

In [46]:

```
1 askint()
```

```
Please enter an integer: five\
Looks like you did not enter an integer!
Finally, I executed!!

---------------------------------------------------------------------------
-
UnboundLocalError                         Traceback (most recent call las
t)
Cell In[46], line 1
----> 1 askint()

Cell In[44], line 9, in askint()
      6 finally:
      7     print('Finally, I executed!!')
----> 9 print(val)

UnboundLocalError: local variable 'val' referenced before assignment
```

توجه کنید که وقتی سعی کردیم val را چاپ کنیم (زیرا به درستی اختصاص داده نشده بود)، خطایی دریافت کردیم. بیایید با پرسیدن کاربر و بررسی اینکه نوع ورودی یک عدد صحیح است، این مشکل را برطرف کنیم:

In [47]:

```
1  def askint():
2      try:
3          val = int(input('Please enter an integer: '))
4      except:
5          print('Looks like you did not enter an integer!')
6          val = int(input('Please agian-enter an integer: '))
7      finally:
8          print('Finally, I executed!!')
9
10     print(val)
```

In [48]:

```
1 askint()
```

```
Please enter an integer: five
Looks like you did not enter an integer!
Please agian-enter an integer: five
Finally, I executed!!

---------------------------------------------------------------------------
-
ValueError                                Traceback (most recent call las
t)
Cell In[47], line 3, in askint()
      2 try:
----> 3     val = int(input('Please enter an integer: '))
      4 except:

ValueError: invalid literal for int() with base 10: 'five'

During handling of the above exception, another exception occurred:

ValueError                                Traceback (most recent call las
t)
Cell In[48], line 1
----> 1 askint()

Cell In[47], line 6, in askint()
      4 except:
      5     print('Looks like you did not enter an integer!')
----> 6     val = int(input('Please agian-enter an integer: '))
      7 finally:
      8     print('Finally, I executed!!')

ValueError: invalid literal for int() with base 10: 'five'
```

هممم... فقط یک بار چک کردیم. چطور می توانیم به طور مداوم بررسی کنیم؟ ما می توانیم از یک حلقه while استفاده کنیم!

In [49]:

```
1  def askint():
2      while True:
3          try:
4              val = int(input('Please enter an integer: '))
5          except:
6              print('Looks like you did not enter an integer!')
7              continue
8          else:
9              print("Yep that's an integer")
10             break
11         finally:
12             print('Finally, I executed!!')
13
14         print(val)
```

In [50]:

```
askint()
```

```
Please enter an integer: five
Looks like you did not enter an integer!
Finally, I executed!!
Please enter an integer: 5
Yep that's an integer
Finally, I executed!!
```

پس چرا تابع ما "بالاخره، من اجرا شدم!" را پس از هر دور چاپ کرد، اما هیچگاه val را چاپ نکرد؟ این به این دلیل است که با یک بلوک try/except/finally، هر دستور continue یا break تا پس *از* اتمام بلوک try رزرو شده‌اند. این بدان معنی است که با وجود اینکه ورودی موفقیت آمیزی به ما آورد **3** وارد بلوک else: شدیم و یک دستور break پرتاب شد، بلوک try تا قبل از خروج از حلقه while ادامه یافت. و از آنجا که print(val) خارج از بلوک try بود، دستور break از اجرای آن جلوگیری کرد.

بیایید یک تنظیم نهایی انجام دهیم:

In [51]:

```
def askint():
    while True:
        try:
            val = int(input('Please enter an integer: '))
        except:
            print('Looks like you did not enter an integer!')
            continue
        else:
            print("Yep that's an integer")
            break
        finally:
            print('Finally, I executed!!')

    print(val)
```

In [52]:

```
askint()
```

```
Please enter an integer: ddsf
Looks like you did not enter an integer!
Finally, I executed!!
Please enter an integer: sdfsf
Looks like you did not enter an integer!
Finally, I executed!!
Please enter an integer: sasda
Looks like you did not enter an integer!
Finally, I executed!!
Please enter an integer: 77
Yep that's an integer
Finally, I executed!!
77
```

عالی! اکنون شما می‌دانید چگونه با استفاده از نمادهای try، except، else و finally، خطاها و استثناءها را در پایتون کنترل کنید!