حل مسایل برنامه‌نویسی پیشرفته با پایتون

تألیف:

دکتر رمضان عباس نژاد ورزی
مهندس محمد نادعلی زاده چاری

فن‌آوری نوین

سرشناسه	:	عباس‌نژاد ورزی، رمضان، ۱۳۴۸ –
عنوان و نام پدیدآور	:	حل مسایل برنامه‌نویسی پیشرفته با پایتون/تألیف رمضان عباس‌نژادورزی، محمد نادعلی‌زاده‌چاری.
مشخصات نشر	:	بابل: فناوری نوین، ۱۳۹۸.
مشخصات ظاهری	:	۲۲۴ ص.: مصور ، جدول .
شابک	:	۴۵۰۰۰۰ ریال:۹۷۸-۶۰۰-۷۲۷۲-۳۷-۴
وضعیت فهرست نویسی	:	فیپا
موضوع	:	پایتون (زبان برنامه‌نویسی کامپیوتر)
موضوع	:	Python (Computer program language)
شناسه افزوده	:	نادعلی‌زاده چاری، محمد، ۱۳۶۴ –
رده بندی کنگره	:	QA۷۶/۷۳
رده بندی دیویی	:	۰۰۵/۱۳۳
شماره کتابشناسی ملی	:	۵۹۲۱۵۱۰

www.fanavarienovin.net

فن‌آوری نوین بابل، کد پستی ۴۷۱۶۷-۷۳۴۴۸ تلفن: ۳۲۲۵۶۶۸۷-۰۱۱

حل مسایل برنامه‌نویسی پیشرفته با پایتون

تألیف: رمضان عباس نژاد ورزی، محمد نادعلی زاده چاری

نوبت چاپ: چاپ اول

سال چاپ: پاییز ۱۳۹۸

شمارگان: ۲۰۰

قیمت: ۴۵۰۰۰ تومان

نام چاپخانه و صحافی: دفتر فنی سورنا

شابک: ۹۷۸-۶۰۰-۷۲۷۲-۳۷-۴

نشانی ناشر: بابل، چهارراه نواب، کاظم بیگی، جنب مسجد منصور کاظم بیگی، طبقه اول

طراح جلد: کانون آگهی و تبلیغات آبان (احمد فرجی)

تهران، خ اردیبهشت، نبش وحید نظری، پلاک ۱۴۲ تلفکس: ۶۶۴۰۰۱۴۴-۶۶۴۰۰۲۲۰

فهرست مطالب

مقدمه

زبان برنامه‌نویسی پایتون کاربردهای متعددی دارد. از طریق این زبان می‌توان برنامه‌های تحت کنسول، برنامه‌های تحت ویندوز (دسک‌تاپی)، برنامه‌های تحت وب، برنامه‌های کار با بانک اطلاعاتی، برنامه‌های تحت شبکه و غیره را پیاده‌سازی نمود. پایتون برای پیاده‌سازی این برنامه‌ها از کتابخانه‌های مختلفی بهره می‌برد. در این کتاب به مطالعه برخی از این کتاب‌خانه‌ها و چگونگی استفاده از آن‌ها می‌پردازیم. قبل از مطالعه این کتاب باید مقدمات برنامه‌نویسی پایتون را یاد بگیرید. برای این منظور می‌توانید سه کتاب به نام‌های **آموزش گام‌به‌گام برنامه‌نویسی پایتون، حل مسائل پایتون** و **آموزش گام‌به‌گام برنامه‌نویسی پیشرفته با پایتون** از همین انتشارات را مطالعه کنید. این کتاب ادامه، مکمل کتاب **آموزش گام‌به‌گام برنامه‌نویسی پیشرفته با پایتون** است. کتاب حاضر شامل ۸ فصل است.

فصل اول، مثال‌های از انواع خطاها و اداره کردن استثنا را ارائه کرده است. **فصل دوم**، شامل مثال‌های از ماژول Turtle است. **فصل سوم**، دارای مثال‌های از ماژول Canvas (برای رسم اشکال گرافیکی مختلف و نمایش تصاویر) می‌باشد. **فصل چهارم**، دارای مثال‌های از بسته Tkinter (این بسته به همراه پایتون نصب می‌شود و برای ایجاد واسط کاربری به کار می‌رود) می‌باشد. **فصل پنجم**، مثال‌های متعددی از برنامه‌نویسی هم‌زمان و پردازش موازی در پایتون دارد. **فصل ششم**، دارای مثال‌های از بسته PyQT (این بسته به همراه پایتون نصب نمی‌شود. پس برنامه‌نویس باید ابتدا آن را نصب کند که برای ایجاد واسط کاربری پیشرفته به کار می‌رود) می‌باشد.در **فصل هفتم**، مثال‌های از برنامه‌نویسی شبکه و استفاده ماژول socket را شامل می‌شود. **فصل هشتم**، یک پروژه از مبحث بانک اطلاعاتی را حل می‌کند.

امیدواریم این کتاب نیز مورد استقبال اساتید و دانشجویان رشته‌های مختلف که زبان پایتون را مطالعه می‌کنند، قرار گیرد.

مؤلف
fanavarienovin@gmail.com

فصل ۱ مدیریت استثناها

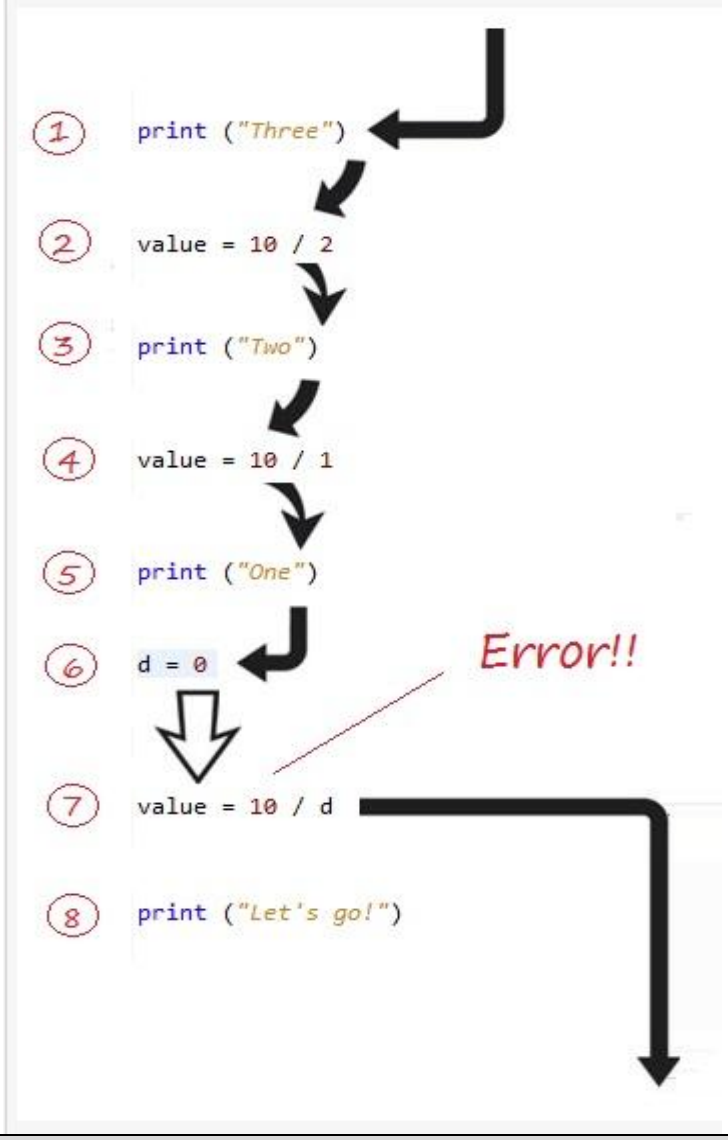
مثال ۱–۱. برنامه‌ای که رخ دادن استثنای تقسیم بر ۰ را بدون کنترل آن نمایش می‌دهد:

```
print ("Three")
# This division no problem.
value = 10 / 2
print ("Two")
# This division no problem.
value = 10 / 1
print ("One")
d = 0
# This division has problem, divided by 0.
# An error has occurred here.
value = 10 / d
# And the following code will not be executed.
print ("Let's go!")
```

دستور اول، "three" را نمایش می‌دهد، دستور دوم، یک توضیح است، دستور سوم، مقدار ۱۰ تقسیم بر ۲ را در value قرار می‌دهد، دستور چهارم، نیز یک توضیح است، دستور پنجم ۱۰ تقسیم بر ۱ را در value قرار می‌دهد. دستور ششم، عبارت "one" را نمایش می‌دهد. دستور هفتم ۰ را در d قرار می‌دهد، دستورات هشتم و نهم، توضیحات هستند، دستور دهم، می‌خواهد ۱۰ تقسیم بر d (یعنی ۰) را در value قرار دهد که یک خطای تقسیم بر صفر اتفاق می‌افتد، فرآیند اجرای برنامه قطع می‌شود و یک پیغام نمایش داده می‌شود (خطای اتفاق افتاده زمان اجرا را نمایش می‌دهد) (مانند خروجی زیر):

```
Three
Two
One
Traceback (most recent call last):
  File "D:/BookCSharp/کتاب‌های در دست تألیف-/حل مسائل پایتون پیشرفته/exception/exception۱.py", line ۱۱, in <module>
    value = ۱۰ / d
ZeroDivisionError: division by zero
```

مراحل اجرای این برنامه در شکل زیر آمده است:



مثال ۲–۱. برنامه‌ای که رخ دادن استثنای تقسیم بر ۰ را با کنترل آن نمایش می‌دهد.

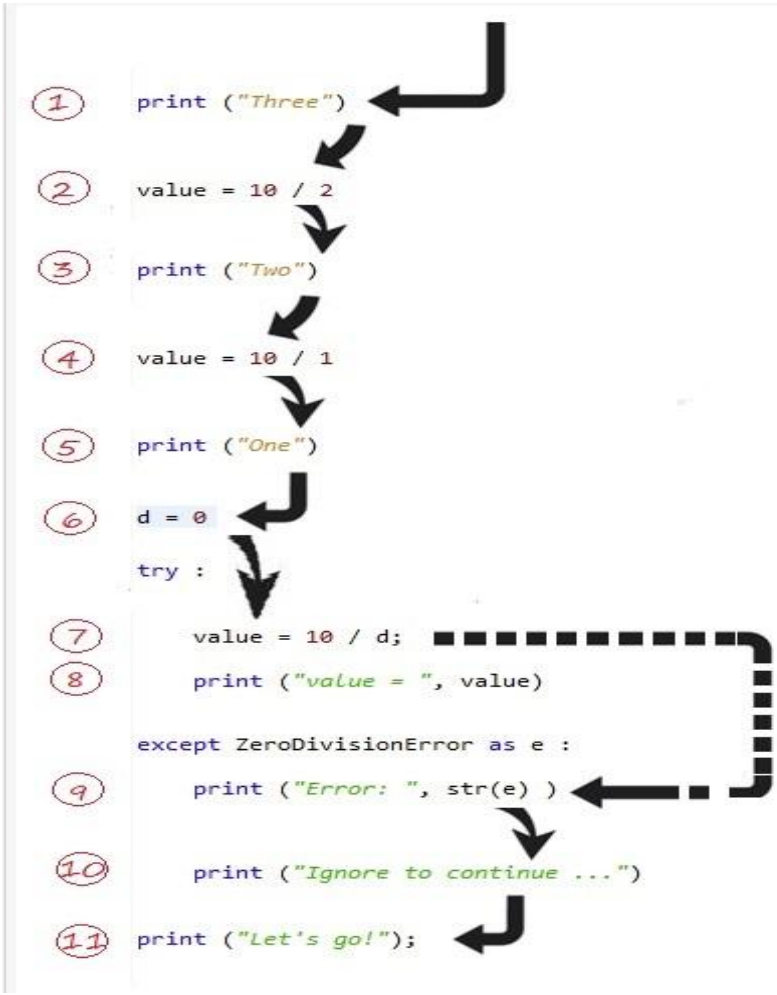
```
print ("Three")
value = 10 / 2
print ("Two")
value = 10 / 1
print ("One")
d = 0
try :
    # This division has problem, divided by 0.
    # An error has occurred here (ZeroDivisionError).
    value = 10 / d
    print ("value = ", value)
```

```
except ZeroDivisionError as e :
    print ("Error: ", str(e) )
    print ("Ignore to continue ...")
print ("Let's go!")
```

این دستورات مانند دستورات مثال قبلی عمل می‌کنند، با این تفاوت که، دستور value = ۱۰/d را در یک بلاک try قرار می‌دهد تا اگر استثنای تقسیم بر صفر در زمان اجرا اتفاق افتاده باشد، دستورات بخش except اجرا شوند و اجرای برنامه قطع نگردد. لذا پیغام "let's go!" نیز نمایش داده می‌شود (مانند خروجی زیر):

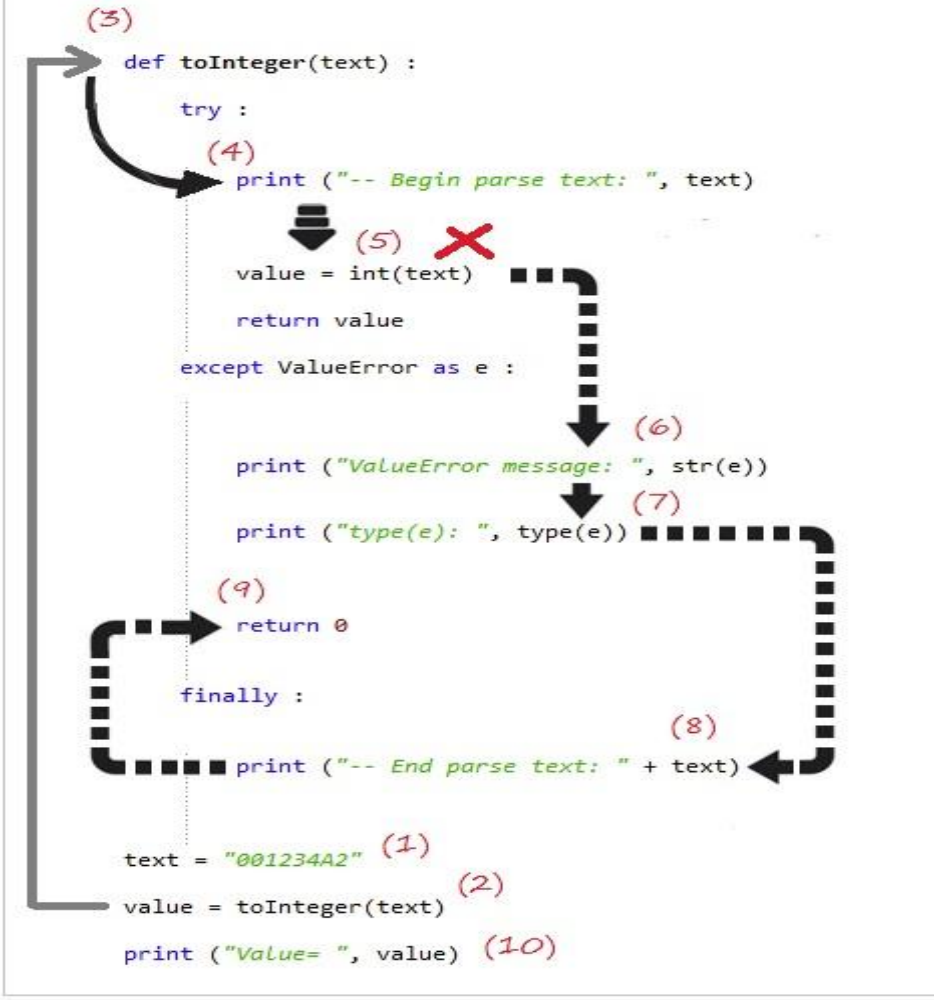
```
Three
Two
One
Error:  division by zero
Ignore to continue ...
Let's go!
```

مراحل اجرای این برنامه در شکل زیر آمده است:



مثال ۳–۴. برنامه‌ای که یک رشته عددی را خوانده، آن را توسط تابعی تبدیل به عدد صحیح می‌کند و نمایش مـــی-دهد. در این برنامه، استثنای ValueError اداره می‌شود.

```
def toInteger(text):
   try :
     print("– Begin parse text:", text)
     value = int(text)
     return value
   except ZeroDivisionError as e :
     print ("ValueError Message: ", str(e) )
     print ("type(e): ", type(e))
     return 0
   finally:
     print("– End parse text:", text)
text = "00123442"
value = toInteger(text)
print("value:", value)
```



دستور اول، تابع toInteger() را تعریف می‌کند که پارامتر text را دریافت کرده، اعمال زیر را انجام می‌دهد:

۱. یک بلاک try ایجاد می‌کند. در این بلاک، ابتدا، با یک پیغام مناسب text را نمایش می‌دهد، سپس توسط تابع int()، می‌خواهد مقدار text را به عدد صحیح تبدیل کند، اگر به عدد صحیح تبدیل شود، آن را در value قرار می‌دهد و value را برمی‌گرداند.
۲. اگر در بلاک try، خطای اتفاق افتد، بخش except اجرا می‌شود و اگر خطای رخ‌داده ValueError باشد، پیغام و نوع خطای اتفاق افتاده را نمایش داده، مقدار صفر را برمی‌گرداند.
۳. درهرصورت (چه بخش try خطای رخ دهد و چه خطای رخ ندهد)، بخش finally یک پیغام مناسب به همراه text را نمایش می‌دهد.

ماژول اصلی، ابتدا با یک پیغام مناسب text را خوانده، سپس با تابع toInteger() آن را به عدد تبدیل کرده و در پایان، value را با پیغام مناسب نمایش می‌دهد.

```
— Begin parse text: ۰۰۱۲۳٤٤۲
— End parse text: ۰۰۱۲۳٤٤۲
value: ۱۲۳٤٤۲
```

برنامه را دوباره اجرا کنید:

```
Enter a input:two
-- Begin parse text:  two
ValueError message:  invalid literal for int() with base ۱۰: 'two'
type(e):  <class 'ValueError'>
-- End parse text: two
Value= ۰
```

مثال ۴–۱. برنامه‌ای که دو عدد را خوانده، عدد اول را بر عدد دوم تقسیم می‌کند. در این برنامه، استثناهای ValueError وZeroDivisionError اداره می‌شوند.

```
number1 = input( "Enter number1: " )
number2 = input( "Enter number2: " )
try:
   number1 = float( number1 )
   number2 = float( number2 )
   result = number1 / number2
except ValueError:
   print ("You must enter two numbers")
except ZeroDivisionError:
   print ("Attempted to divide by zero")
else:
   print ("%.3f / %.3f = %.3f" % ( number1, number2, result ))
```

دستورات اول و دوم، با پیغام‌های مناسب دو عدد را می‌خوانند، دستور سوم، یک بلاک try ایجاد می‌کند، در این بلاک try، ابتدا اعداد اول و دوم خوانده‌شده را با متد float() به عدد اعشاری تبدیل می‌کنند و در number و number۲ قرار می‌دهند، سپس number تقسیم بر number۲ را در result قرار می‌دهند. اگر در

این بلاک خطای ValueError اتفاق افتد، پیغام You must enter two numbers را نمایش می‌دهد، امّا، اگر خطای ZeroDivisionError رخ دهد، پیغام Attempted to divide by zero را نمایش می‌دهد، ولی اگر هیچ خطای رخ ندهد، عدد اول، دوم و حاصل تقسیم آن‌ها را نمایش می‌دهد.

```
Enter number۱: ۱۰
Enter number۲: ۱۲
۱۰,۰۰۰ / ۱۲,۰۰۰ = ۰,۸۳۳
```

برنامه را دوباره اجرا کنید:

```
Enter number۱: ۲۰
Enter number۲: ۰
Attempted to divide by zero
```

برنامه را مجدداً اجرا کنید:

```
Enter number۱: One
Enter number۲: Three
You must enter two numbers
```

مثال ۵–۱. برنامه‌ای که مقادیر یک لیست را به نوع عددی اعشاری تبدیل کرده و نمایش می‌دهد. در این برنامه، استثناهای ValueError وTypeError اداره می‌شوند.

```
for value in ("18.0", None, "Hi!", "12.0"):
    try:
        print ("Attempting to convert", value, "-->",)
        print (float(value))
    except(TypeError):
        print ("I can only convert a string or a number!")
    except(ValueError):
        print ("I can only convert a string of digits!")
```

دستور اول، مقادیر یک لیست را با حلقه for پیمایش می‌کند، دستور دوم، یک بلاک try را تعریف می‌نماید که در این بلاک، ابتدا یک پیغام مناسب نمایش داده می‌شود، سپس عضو فعلی لیست که در value قرار دارد را به عدد اعشاری تبدیل می‌کند و نمایش می‌دهد. در هنگام تبدیل اگر خطای TypeError رخ دهد، پیغام I can only convert a string or a number! را نمایش می‌دهد. امّا، اگر خطای ValueError اتفاق بی‌افتد، پیغام I can only convert a string of digits! را نمایش خواهد داد.

```
Attempting to convert ۱۸,۰ -->
۱۸,۰
Attempting to convert None -->
I can only convert a string or a number!
Attempting to convert Hi! -->
I can only convert a string of digits!
Attempting to convert ۱۲,۰ -->
۱۲,۰
```

مثال ۶–۱. برنامه‌ای که یک استثنا را پرتاب می‌نماید.

```
def raiseExceptionDoNotCatch():
   try:
      print ("In raiseExceptionDoNotCatch")
      raise Exception
   finally:
      print ("Finally executed in raiseExceptionDoNotCatch")
   print ("Will never reach this point")
print ("\nCalling raiseExceptionDoNotCatch")
try:
   raiseExceptionDoNotCatch()
except Exception:
   print ("Caught exception from raiseExceptionDoNotCatch in main
program.")
```

دستور اول، تابع raiseExceptionDoNotCatch() را تعریف می‌کند که یک بلاک try ایجاد می‌نماید. در این بلاک ابتدا، یک پیغام نمایش داده و سپس با کمک کلیدی raise یک استثنا را تولید می‌کند. در بخش finally یک پیغام دیگر نمایش داده می‌شود. در ادامه، دو پیغام را نمایش می‌دهد و بلاک try جدید ایجاد می‌نماید. در بخش try، تابع raiseExceptionDoNotCatch() را فراخوانی می‌کند و در بخش except چنانچه خطای رخ دهد، پیغام مناسبی را نمایش خواهد داد.

```
Calling raiseExceptionDoNotCatch
In raiseExceptionDoNotCatch
Finally executed in raiseExceptionDoNotCatch
Caught exception from raiseExceptionDoNotCatch in main program.
```

مثال ۷–۱. برنامه‌ای دو عدد و یک رشته را خوانده، عدد اول را بر عدد دوم تقسیم می‌کند و نمایش می‌دهد و کاراکتر اندیس ۸ رشته را نمایش می‌دهد. در این برنامه، با توجه به رخ دادن استثناهای IndexError و ZeroDivitionError پیغام مناسب نمایش می‌دهد.

```
try:
    x = int(input("enter 1st number: "))
    y = int(input("enter 2nd number: "))
    print(x/y)
    string=input("enter some string: ")
    print(string[8])
except (IndexError, ZeroDivisionError) as e:
    print("An error occurred :",e)
else:
    print("no error")
```

دستور اول، یک بلاک try را ایجاد می‌کند. در این بلاک ابتدا، دو عدد را با پیغام‌های مناسب می‌خواند، سپس حاصل تقسیم عدد اول بر عدد دوم را نمایش می‌دهد. در ادامه، با یک پیغام مناسب یک رشته را خوانده، کاراکتر اندیس ۱ رشته را نمایش می‌دهد. اگر در دستورات بلاک try یکی از استثنای تقسیم بر صفر، خارج از اندیس رخ دهد، در بخش except، با پیغام مناسب آن خطا را نمایش می‌دهد (در این برنامه یک except وجود دارد که e (یعنی، خطای رخ‌داده) را نمایش می‌دهد. امّا، اگر خطای اتفاق نی‌افتد، در پایان، «no error» را نمایش می‌دهد.

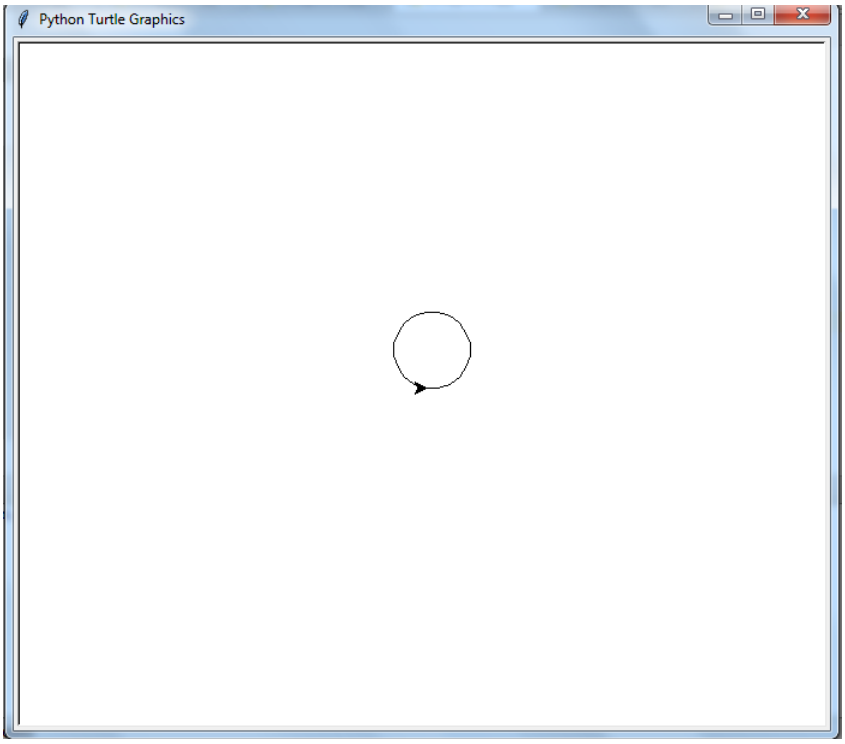
```
enter ۱st number: ۱۲
enter ۲nd number: ۱٤
۰,۸۵۷۱٤۲۸۵۷۱٤۲۸۵۷۱
enter some string: ali
An error occurred : string index out of range
```

برنامه را مجدداً اجرا کنید:

```
enter ۱st number: ۱۰
enter ۲nd number: ۰
An error occurred : division by zero
```

مثال ۸–۱. برنامه‌ای که توسط یک تابع یک دایره را در محیط turtle رسم می‌کند. اگر در هنگام رسم خطای رخ دهد پیغام Error را نمایش می‌دهد.

```
import turtle
import time
def showCircle(n):
    try:
        win = turtle.Screen()
        t = turtle.Turtle()
        angle = 360 / n
        for i in range(n):      # Draw the polygon
            t.forward(10)
            t.left(angle)
        time.sleep(3)           # Make program wait a few seconds
    except:
        print("Error")
showCircle(20)
showCircle(0)
```



دستوارت اول و دوم، ماژول‌های موردنیاز را به برنامه اضافه می‌کنند، دستور سوم، متد showCirclec() را پیاده‌سازی می‌نماید. در این متد، یک بلاک try وجود دارد. در این بلاک، ابتدا صفحه turtle را ایجاد کرده و توسط یک حلقه for، دایره را در آن رسم می‌کند، در پایان، ۳ میلی‌ثانیه مکث می‌نماید. چنانچه در داخل بلاک try خطای رخ دهد، پیغام Error را نمایش می‌دهد.

در ماژول اصلی دو بار showCircle () را با مقادیر پارامترهای ۲۰ و ۰ فراخوانی می‌کند.

Error

مثال ۹–۱. برنامه‌ای که نام یک فایل را خوانده، اطلاعات آن فایل را نمایش می‌دهد. چنانچه فایل وجود نداشته باشد، استثنای رخ‌داده و خطای اتفاق می‌افتد.

```
import sys
text = []
try:
    fileName=input("Enter a file name:")
    fh = open(fileName, 'r')
    text = fh.readlines()
    fh.close()
except IOError:
    print('cannot open', fileName)
if text:
    print(text)
```

دستور اول، ماژول sys را به برنامه اضافه می‌کند، دستور دوم، لیست text را به‌صورت خالی تعریف می‌نماید. دستور سوم، یک بلاک try را ایجاد کرده، در این بلاک، ابتدا با پیغام مناسبی نام یک فایل را

خوانده، فایل را به‌صورت فقط خواندنی باز می‌کند و اطلاعات آن را می‌خواند و در text قرار می‌دهد. در پایان، فایل را می‌بندد. چنانچه در بلاک try خطای IoError رخ‌داده باشد، پیغام مناسبی را نمایش می‌دهد.

در انتهای برنامه، اگر text خالی نباشد، اطلاعات آن را نمایش می‌دهد.

```
Enter a file name:d:/data/۲.txt
['Java\n', 'Cpp\n', 'Cobol\n', 'Python\n', 'C#\n', 'Php\n']
```

برنامه را مجدداً اجرا کنید:

```
Enter a file name:d:/۱۱۱۱۱.txt
cannot open d:/۱۱۱۱۱.txt
```

مثال ۱۰–۱. برنامه‌ای که از طریق کلاس چند استثنا را تعریف کرده و توسط برنامه اصلی آن‌ها را چک می‌کند.

```
class Error(Exception):
   """Base class for other exceptions"""
   pass
#Define class for NegativeValueError
class NegativeValueError(Error):
  """Raised when the input is negative"""
  pass
#Define class for ValueTooSmallError
class ValueTooSmallError(Error):
   """Raised when the value is too small"""
   pass
#Define class for ValueTooLargeError
class ValueTooLargeError(Error):
   """Raised when the value is too large"""
   pass
number = 11
while True:
   try:
       num = int(input("Enter a number: "))
       if num < 0:
           raise NegativeValueError
       elif num < number:
           raise ValueTooSmallError
       elif num > number:
           raise ValueTooLargeError
       break
   except NegativeValueError:
       print("This is a negative value, try again")
       print("")
   except ValueTooSmallError:
       print("This value is too small, try again")
       print("")
   except ValueTooLargeError:
       print("This value is too large, try again!")
       print("")
print("Correct value entered")
```

دستور اول، کلاس Error را تعریف می‌کند که مشتق کلاس Exception است، دستور دوم، کلاس NegativeValueError را تعریف می‌کند که مشتق کلاس Error است، دستور سوم، کلاس

ValueTooSmallError را تعریف می‌کند که مشتق کلاس Error می‌باشد، دستور چهارم، کلاس ValueTooLargeError را تعریف می‌کند که مشتق کلاس Error است.

در ماژول اصلی، ابتدا متغیر number را برابر ۱۱ قرار می‌دهد و یک حلقه while بی‌نهایت ایجاد می‌کند. در این حلقه اعمال زیر را انجام می‌شود:

۱. یک عدد خوانده در num قرار می‌دهد.

۲. اگر num<۰ باشد، استثنایNegativeValueError را پرتاب می‌کند، اگر num۴<number باشد استثنای ValueTooSmallError را پرتاب می‌نماید، اما اگر num>number باشد، استثنای ValueTooLargeError را پرتاب خواهد کرد و از حلقه خارج می‌شود. در بخش except مربوط به NegativeValueError، پیغام "This is a negative vglue, try again!" نمایش داده می‌شود، اما در بخش except مربوط به ValueTooSmallError، پیغام " This Value is too small, try again!" نمایش داده خواهد شد، ولی در بخش except مربوط به ValueTooLargeError، پیغام " This value is to large, try agan!" را نمایش می‌دهد. "correct value entered" را نمایش می‌دهد.

```
Enter a number: ۱۰
This value is too small, try again

Enter a number: ۱۷
This value is too large, try again!

Enter a number: -۸
This is a negative value, try again

Enter a number:
```

مثال ۱۱–۱. برنامه‌ای که یک سال را دریافت کرده، تعیین می‌کند که کبیسه است یا خیر. در این برنامه، ValueError اداره می‌شود.

```
try:
    Year = int(input("Please Enter a Year? "))
    leap_or = Year%4 == 0
    if leap_or:
        print ("This is a leap year")
    else:
        print ("This is not a leap year")
except ValueError:
    print ("Only digits are allowed!")
```

دستور اول، بلاک try را ایجاد می‌کند، در این بلاک ابتدا، با یک پیغام مناسب سال را خوانده، در متغیر Year قرار می‌دهد. سپس نتیجه باقی‌مانده تقسیم Year به ۴ را در leab_or قرار می‌دهد. اگر leap_or برابر True باشد، پیغام "This is a leap year" را نمایش می‌دهد. وگرنه پیغام "This is not a leap year" را نمایش خواهد داد. اگر در دستورات بلاک try استثنای ValueError رخ دهد، پیغام " Only digits are allowed" را نمایش می‌دهد.

```
Please Enter a Year? ۲۰۲۰
This is a leap year
```

برنامه را مجدداً اجرا کنید:

```
Please Enter a Year? ۲۰To۲۰
Only digits are allowed!
```

مثال ۱۲–۱. برنامه‌ای که یک عدد را خوانده، این عدد را بر تمام اعضای یک لیست موجود تقسیم می‌نماید. در این برنامه، استثناهای ZeroDivisionError و TypeError مدیریت می‌شوند.

```
num_list = [5, 'c', 20, 0, 40]
a = int(input("Enter a number:"))
for b in num_list:
    try:
        c = a / b
        print(c)
    except ZeroDivisionError:
         print("An element with 0 value")
    except TypeError:
        print("Only numbers can be used")
```

دستور اول، لیست num_list را با مقادیر تعیین‌شده تعریف می‌کند، دستور دوم، با یک پیغام مناسب عددی را خوانده در a قرار می‌دهد، دستور سوم، یک حلقه for ایجاد کرده که اعضای لیست را پیمایش می‌نماید. در این حلقه، یک بلاک try ایجاد می‌نماید که a را بر اعضای لیست تقسیم کرده، در c قرار می‌دهد وc را نمایش می‌دهد. اگر در بلاک try خطای ZeroDivisionError رخ دهد، پیغام "An element with ۰ value" نمایش داده می‌شود. ولی اگر استثنای TyrpeError اتفاق بی افتد، پیغام "Only numbers can be used" نمایش داده خواهد شد.

```
Enter a number:۲۰
٤,۰
Only numbers can be used
۱,۰
An element with ۰ value
۰,۵
```

مثال ۱۳–۱. برنامه‌ای که مشخصات یک شخص از قبیل نام، نام خانوادگی، سن، قد و وزن را دریافت می‌کند. در این برنامه، چنانچه در هنگام دریافت مشخصات استثنای رخ دهد آن را مدیریت خواهد کرد.

```
person = {}
properties = [
    ("name", str),
    ("surname", str),
    ("age", int),
    ("height", float),
    ("weight", float),
]
for property, p_type in properties:
    valid_value = None
    while valid_value is None:
        try:
            value = input("Please enter your %s: " % property)
```

```
            valid_value = p_type(value)
        except ValueError as ve:
            print(ve)
    person[property] = valid_value
```

دستور اول، مجموعه person را به‌صورت خالی تعریف می‌کند، دستور دوم، ویژگی‌های شخص (خواص شخص) را به‌صورت دیکشنری تعریف می‌کند که فیلد کلید نام و فیلد مقدار نوع آن ویژگی را تعیین می‌کند. دستور سوم، یک حلقه for ایجاد کرده که تمام ویژگی‌های شخص را پیمایش می‌کند. در داخل این حلقه، ابتدا valid_value را برابر None قرار می‌دهد و تا زمانی که valid_value برابر None باشد، یک حلقه جدید دیگر را تکرار می‌کند، در داخل این حلقه یک بلاک try ایجاد می‌نماید. در این بلاک یک پیغام مناسب نمایش می‌دهد که یک ویژگی را خوانده و آن ویژگی را به نوع خودش تبدیل می‌کند.

چنانچه در هنگام دریافت ویژگی‌ها استثنای ValueError رخ دهد، پیغام مناسبی را نمایش می‌دهد.

```
Please enter your name: Ali
Please enter your surname: Abbasi
Please enter your age: ۲five
invalid literal for int() with base ۱۰: '۲five'
Please enter your age: ۲۵
Please enter your height: ۳۰H
could not convert string to float: '۳۰H'
Please enter your height: ۳۰
Please enter your weight: ۱۲
```

مثال ۱۴–۱. برنامه‌ای که از طریق کلاس چند استثنا را پیاده‌سازی کرده، آن‌ها را آزمایش می‌نماید.

```
class GenderException(Exception):
    def __init__(self, msg):
        super().__init__(msg)
class LanguageException(Exception):
    def __init__(self, msg):
        super().__init__(msg)
class PersonException(Exception):
    def __init__(self, msg):
        super().__init__(msg)
def checkGender(gender):
    if gender != 'Female':
        raise GenderException("Accept female only")
def checkLanguage(language):
    if language != 'English':
        raise LanguageException("Accept english language only")
def checkPerson(name, gender, language):
    try:
        # May throw GenderException.
        checkGender(gender)
        # May throw LanguageException.
        checkLanguage(language)
    except Exception as e:
        raise PersonException(name + " does not pass") from e
```

```
try:
    checkPerson("Nguyen", "Female", "Vietnamese")
except PersonException as e:
    print("Error message: ", str(e))
    # GenderException or LanguageException
    cause=e.__cause__
    print('e.__cause__: ', repr(cause))
    print("type(cause): ", type(cause))
    print(" ------------------ ")
    if type(cause) is GenderException:
        print("Gender exception: ", cause)
    elif type(cause) is LanguageException:
        print("Language exception: ", cause)
```

دستور اول، کلاس GenderException را پیاده‌سازی می‌کند که مشتق کلاس Exception است. این کلاس متد ()__init__ را پیاده‌سازی می‌کند که سازنده کلاس پدرش را فراخوانی می‌نماید.

دستور دوم، کلاس LanguageException را پیاده‌سازی می‌کند که مشتق کلاس Exception است. این کلاس دارای یک متد ()__init__ است که سازنده کلاش پدرش را فراخوانی می‌کند.

دستور سوم، کلاس PersonException را تعریف می‌کند که مشتق کلاس Exception است. این کلاس نیز سازنده ()__init__ را پیاده‌سازی می‌کند که سازنده کلاس پدرش را فراخوانی می‌کند.

دستور چهارم، تابع ()checkGender را تعریف کرده که پارامتر gender را دریافت می‌کند، اگر gender مخالف 'Female' باشد، یک استثنای GenderException را فراخوانی می‌کند که پارامتر msg آن "Accept female only" است.

دستور پنجم، تابع ()checkLanguage را تعریف می‌کند که پارامتر language را دریافت می‌کند. اگر پارامتر language مخالف "English" باشد، استثنای LanguageException را با پارامتر " Accept English language only" پرتاب می‌کند.

دستور ششم، تابع ()checkPerson را تعریف می‌کند که پارامترهای name، gender و language را دریافت کرده، در یک بلاک try توابع ()checkGender و ()checkLanguage را فراخوانی می‌کند. اگر در داخل بلاک try استثنای رخ دهد، پیغام مناسب را نمایش می‌دهد.

در ادامه، ماژول اصلی شروع می‌شود. در این ماژول یک try ایجاد کرده است که در داخل بلاک آن تابع ()checkPerson را فراخوانی می‌کند. چنانچه در داخل این بلاک استثنای رخ دهد، در بخش except پیغام مناسب نمایش خواهد داد.

```
Error message:  Nguyen does not pass
e.__cause__:  LanguageException('Accept english language only',)
type(cause):  <class '__main__.LanguageException'>
 ------------------
Language exception:  Accept english language only
```

مثال ۱۵–۱. برنامه‌ای که در یک دیکشنری استثنای KeyError و استثناهای دیگر را چک می‌کند

```
zoo = {'penguins': 5, 'lions': 12, 'zebras': 1, 'rhinos': 2}
try:
    rhino_count = zoo['rhinos']
```

```
    print(f'There are {rhoni_count} rhinos in our zoo!')
except KeyError:
    print('There are NO rhinos in our zoo!')
except Exception as e:
    print(f'Got exception of type {type(e)}: {e}')
    print("Not sure what happened, so it's not safe to continue --
crashing the script!")
    raise e
```

دستور اول، یک دیکشنری به نام zoo با مقادیر تعیین‌شده تعریف می‌کند، دستور دوم، یک بلاک try ایجاد می‌نماید. در این بلاک، مقدار کلیه 'rhinos' را در rhino_count قرار می‌دهد و با یک پیغام مناسب آن را نمایش می‌دهد. اگر در دستورات بلاک try، خطای KeyError رخ دهد، پیغام " There are No rhinos in your zoo" را نمایش می‌دهد. اما اگر استثناهای دیگر رخ دهد، پیغام مناسبی را نمایش می‌دهد و آن استثنا را با دستور raise پرتاب می‌کند.

```
Got exception of type <class 'NameError'>: name 'rhoni_count' is not defined
Not sure what happened, so it's not safe to continue -- crashing the script!
Traceback (most recent call last):
  File "D:/BookCSharp/کتاب‌های در دست تألیف-/حل مسائل پایتون
پیشرفته/exception/exception١٦.py", line ١٠, in <module>
    raise e
  File "D:/BookCSharp/کتاب‌های در دست تألیف-/حل مسائل پایتون
پیشرفته/exception/exception١٦.py", line ٤, in <module>
    print(f'There are {rhoni_count} rhinos in our zoo!')
NameError: name 'rhoni_count' is not defined
```

مثال ۱۶–۱. برنامه‌ای که داده‌های json را بارگذاری می‌کند. در این برنامه، چنانچه خطای اتفاق بی‌افتد، آن را مدیریت می‌نماید.

```
import os
import json
def load_project_data(path):
    data_dict = json.load(open(path))
    print(f"Project loaded: {data_dict['project name']}")
    return data_dict
try:
    data = load_project_data('bogus_data.json')
except Exception as e:
    print(f"Failed to load data: {e}")
else:
    print(data)
```

دستورات اول و دوم، به ترتیب ماژول‌های os و json را به برنامه اضافه می‌کنند، دستور سوم، تابع load_project_data() را تعریف می‌کند که مسیر فایل json به نام path را دریافت می‌نماید و با متدهای load() و open() اطلاعات فایل موجود در مسیر path را در data_dict بار می‌کند، پیغام مناسبی را نمایش می‌دهد و data_dict را برمی‌گرداند.

در برنامه اصلی یک بلاک try ایجاد می‌کند. در این بلاک تابع load_project_data() را فراخوانی می‌کند. اگر در دستورات بلاک try (دستورات تابع فراخوانی شده)، استثنای رخ دهد، پیغام " Failed to load data" را به همراه استثنای رخ‌داده نمایش می‌دهد.

```
Failed to load data: [Errno ۲] No such file or directory: 'bogus_data.json'
```

مثال ۱۷–۱. برنامه‌ای که از طریق کلاس محاسبه تقسیم را انجام می‌دهد. در این برنامه، استثناهای ZeroDivisionError، ValueError و UnicodeError اداره می‌شود.

```
class Calculate:
    def __init__(self):
        self.memory = 0
    def division(self, lhs, rhs):
        try:
            result = float(lhs)/float(rhs)
        except ZeroDivisionError:
            result = "Infinity"
        except (ValueError, UnicodeError):
            print("Oy! Don't diss the sonic!")
            result = "Cannot Calculate"
        else:
            self.memory = result
        finally:
            print(f"Calculation Result: {result}\n")
sonic = Calculate()
sonic.division(8, 4)
sonic.division(4, 0)
sonic.division(4, "zero")
print(f"Memory Is: {sonic.memory}")
```

دستور اول، کلاس Calcuate را تعریف می‌کند، در این کلاس دو متد پیاده‌سازی شده‌اند که عبارت‌اند از:

- متد __nit__()، سازنده کلاس را پیاده‌سازی می‌کند که مقدار فیلد memory را برابر ۰ قرار می‌دهد.
- متد division ()، پارامترهای lhs و rhs را دریافت کرده، در یک بلاک float (lhs) را بر float (rhs) تقسیم کرده، در result قرار می‌دهد. در این بلاک، اگر استثنای ZeroDivisionError رخ دهد، result برابر "Infinity" خواهد شد. اما، اگر یکی از استثناهای ValueError و UnicodeError اتفاق می‌افتد، یک پیغام مناسب نمایش داده و رشته 'Cannot Galcuate' را در result قرار می‌دهد. ولی اگر هیچ استثنایی رخ ندهد، result را فیلد memory قرار می‌دهد.

در ماژول اصلی یک نمونه از کلاس Calculate به نام sonic ایجاد کرده و چندین مرتبه متد division را بر روی آن فراخوانی می‌کند.

```
Calculation Result: ۲,۰

Calculation Result: Infinity
```