دوره جامع برنامه نویسی پایتون

سر فصل مطالب :

1. معرفی و نصب ابزار ها
2. انواع داده ها، تعریف متغیر و اصول نامگذاری، ساختارهای داده (مفاهیم پایه)
3. عملگرهای مقایسه ای و منطقی
4. دستوالعمل ها
5. متد ها و توابع
6. برنامه نویسی شیء گرا
7. ماژول ها و پکیج ها
8. خطاها و مدیریت استثناء ها
9. Decorators and Generators
10. ماژول های پیشرفته
11. مطالب تکمیلی و پیشرفته
12. پروژه ها

فصل ۸

خطاها و مدیریت استثناءها

- **ساختار** try/except/finally
- Unit testing
- Pylint
- Unittest

ساختار try/except/finally

- قطعه کد زیر را در نظر بگیرید، چه خطایی در آن مشاهده میکنید؟

```
print('python)
```

- به نظر شما این چه نوع خطایی است؟
- این نوع خطا به عنوان یک استثنا شناخته می شود. حتی اگر یک عبارت از نظر نحوی صحیح باشد، ممکن است هنگام تلاش برای اجرای آن خطا ایجاد کند. خطاهایی که در حین اجرا شناسایی می شوند استثنا نامیده می شوند و مخرب نیستند
- می توانید لیست کامل استثناهای داخلی را در اینجا بررسی کنید. در این بخش میخواهیم یاد بگیریم که چگونه خطاها و استثناها را در کد خود مدیریت کنیم

ساختار try/except/finally

- Syntax اصلی مورد استفاده برای رسیدگی به خطاها در پایتون عبارت try و except میباشد
- کدی که می تواند باعث ایجاد یک استثنا شود در بلوک try قرار داده می شود و سپس مدیریت استثنا در بلوک except پیاده سازی می شود

```
try:
    You do your operations here...
    ...
except ExceptionI:
    If there is ExceptionI, then execute this block.
except ExceptionII:
    If there is ExceptionII, then execute this block.
    ...
else:
    If there is no exception then execute this block.
```

ساختار try/except/finally

try/except ex1

```
try:
    f = open('testfile.txt','w')
    f.write('Test write this')
except IOError:
    # This will only check for an IOError exception and then execute this print statement
    print("Error: Could not find file or read data")
else:
    print("Content written successfully")
    f.close()
```

ساختار try/except/finally

try/except ex2

```
try:
    f = open('testfile.txt','r')
    f.write('Test write this')
except IOError:
    # This will only check for an IOError exception and then execute this print statement
    print("Error: Could not find file or read data")
else:
    print("Content written successfully")
    f.close()
```

ساختار try/except/finally

try/except ex3

```
try:
    f = open('testfile','r')
    f.write('Test write this')
except:
    # This will check for any exception and then execute this print statement
    print("Error: Could not find file or read data")
else:
    print("Content written successfully")
    f.close()
```

ساختار try/except/finally

- finally
- بلوک کد finally همیشه بدون در نظر گرفتن اینکه استثنایی در بلوک کد try وجود داشته باشد اجرا می شود

try/finally

```
try:
    #Code block here
    #...
    #Due to any exception, this code may be skipped!
finally:
    #This code block would always be executed.
```

ساختار try/except/finally

try/finally ex

```
try:
    f = open("testfile.txt", "w")
    f.write("Test write statement")
    f.close()
except:
    print("Error: Could not find file or read data")
finally:
    print("Always execute finally code blocks")
```

- به همان اندازه که نوشتن کد خوب مهم است، نوشتن تست های خوب نیز مهم است. بهتر است خودتان باگ‌ها را پیدا کنید تا اینکه کاربران نهایی آن‌ها را به شما گزارش کنند!
- کتابخانه ها و پکیج های خوب زیادی برای تست نویسی در پایتون وجود دارد که در زیر به برخی از آن ها اشاره شده

pylint pyflakes pep8

- اینها ابزارهای ساده ای هستند که صرفاً به کد شما نگاه می کنند و به شما می گویند که آیا مشکلات نحو یا مشکلات ساده ای مانند نام متغیرها قبل از تخصیص فراخوانی شده است یا خیر
- یک راه به مراتب بهتر برای آزمایش کد، نوشتن تست هایی است که داده های نمونه را به برنامه شما ارسال می کند و آنچه را که به یک نتیجه دلخواه برمی گردد، مقایسه کنید.
- دو ابزار زیر از کتابخانه استاندارد برای این منظور هستند:

unittest doctest

- برای نصب pylint دستور زیر را در cmd یا terminal وارد کنید:

```
pip install pylint
```

- یا دستور زیر را در jupyter وارد کنید:

```
! pip install pylint
```

- برای استفاده از pylint باید دستور زیر را در cmd وارد کنید:

```
pylint filename.py
```

pylint test 1

```
%%writefile simple1.py
a = 1
b = 2
print(a)
print(B)
```

pylint test 2

```
%%writefile simple1.py
"""
A very simple script.
"""


def myfunc():
    """
    An extremely simple function.
    """
    first = 1
    second = 2
    print(first)
    print(second)

myfunc()
```

- Unittest به شما امکان می دهد برنامه های تست خود را بنویسید. هدف این است که مجموعه خاصی از داده ها را به برنامه خود بفرستید و نتایج برگشتی را در برابر یک نتیجه مورد انتظار تجزیه و تحلیل کنید
- برای افزودن unittest به برنامه باید این کتابخانه را به برنامه خود معرفی کنیم

```
import unittest
```

- برای اجرای تست دستور زیر را وارد میکنیم

```
! python testfile.py
```

unittest 1

```
%%writefile cap.py
def cap_text(text):
    return text.capitalize()
```

unittest 2

```
%%writefile test_cap.py
import unittest
import cap

class TestCap(unittest.TestCase):

    def test_one_word(self):
        text = 'python'
        result = cap.cap_text(text)
        self.assertEqual(result, 'Python')

    def test_multiple_words(self):
        text = 'python language'
        result = cap.cap_text(text)
        self.assertEqual(result, 'Monty Language')

if __name__ == '__main__':
    unittest.main()
```

unittest 3

```
%%writefile cap.py
def cap_text(text):
    return text.title()
```