Python Fundamentals day 61

Today's Agenda

- Exception handling

Exception handling

In computing and computer programming, exception handling is the process of responding to the occurrence of exceptions – anomalous or exceptional conditions requiring special processing - during the execution of a program.

To understand in a better way let us take non-technical example first

Consider the following sentence:

A apple a day keep a doctor away.

Clearly we can see some grammatical mistakes in the above sentence. The correct form of sentence is "An apple a day keeps the doctor away". This follows all the set of rules which English language follows. Similar to these grammatical mistakes in programming language one can commit some syntactical or logical mistakes.

In python we have set of tokens (identifiers, operators, delimiters, keywords). All commands in python are combination of these tokens.

Tokens
Identifiers: A B C D E F G H I J K L M N O P Q R S T U V W X Y Z a b c d e f g h I j k l m n o p q r s t u v w x y z 0 1 2 3 4 5 6 7 8 9
Operators: + - * / % ** // << >> & | ^ ~ < <= > >= <> != ==
Delimiters: ( ) { } [ ] , : . = ; += -= *= /= //= %= &= |= ^= >>= <<= **=
Keywords: and del for is raise assert elif from lambda return break else global not try class except if or while continue exec import pass with def finally in print yield

Consider the following statement

```
for i in range(6):
    print(i)
```

Above statement is correct according to the rules of python. Let us take another statement

```
fun(x):
    print(x)
```

The above statement is invalid as it is not following the syntax of a normal function. The above statement will cause syntax error which can be fixed by following the rule and adding def keyword before the fun(x).

But this is not exception. The mistakes occurred due to syntax are called syntactical errors. But if the mistake is with the logic then it is called as an exception. These exceptions occur during the execution of the program and once an exception is encountered program gets abruptly terminated. To make sure it doesn't get abruptly terminated we have to handle these exceptions.

For eg:

```
a=10+0
b=10-0
c=10*0
d=10/0
print(a,b,c,d)
```

Output:

```
In [1]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test1.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
Traceback (most recent call last):

  File "<ipython-input-1-33651a4fccdf>", line 1, in
<module>
    runfile('C:/Users/rooman/OneDrive/Desktop/python/
test1.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')

  File "C:\Users\rooman\Anaconda3\lib\site-packages
\spyder_kernels\customize\spydercustomize.py", line 827,
in runfile
    execfile(filename, namespace)

  File "C:\Users\rooman\Anaconda3\lib\site-packages
\spyder_kernels\customize\spydercustomize.py", line 110,
in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "C:/Users/rooman/OneDrive/Desktop/python/test1.py",
line 4, in <module>
    d=10/0

ZeroDivisionError: division by zero
```

Above code is perfectly fine with the syntax. But logic is not correct. Hence we get a ZeroDivionError which is one of the exceptions.

$$\begin{tikzcd}
 & \textbf{Errors} \arrow[dl] \arrow[dr] & \\
\textbf{Syntax Errors} & & \textbf{Exceptions}
\end{tikzcd}$$

Syntax Errors	Exceptions
• Syntax mistakes	• Logical mistakes
• Is easily identified by the interpreter	• Is not identified by the interpreter
• Occurs before runtime	• Occurs during runtime

Let us start by considering an example of banking

```
print('Secure connection has been established to bank server')

p=int(input('Enter your principal amount'))
t=int(input('Enter the duration'))
r=10
si=(p*t*r)/100
print('Simple interest =',si)
print('Secure connection has been closed to the bank server')
```

Output:

```
In [2]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test1.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
Secure connection has been established to bank server

Enter your principal amount1000000

Enter the duration2
Simple interest = 200000.0
Secure connection has been closed to the bank server
```

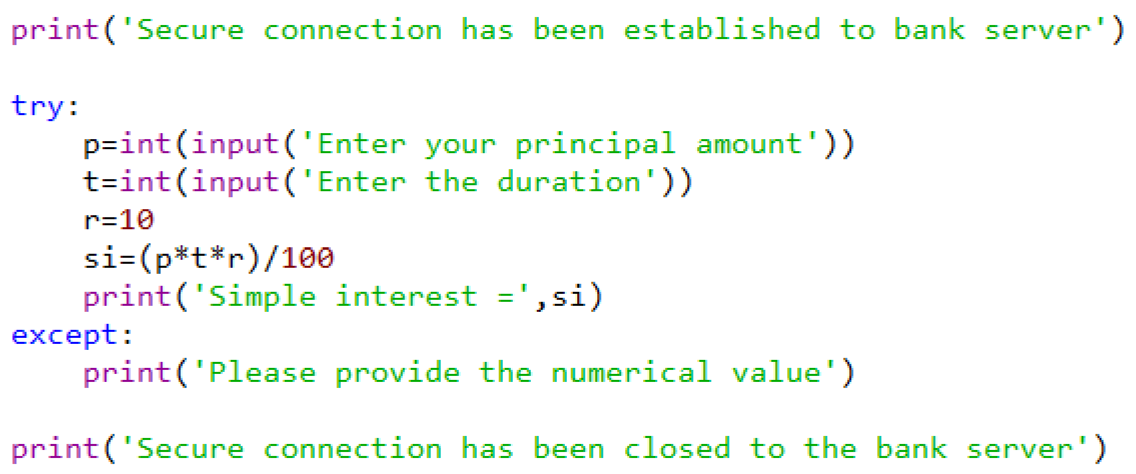
The above code is working fine, as no logical errors occur during execution. But imagine if the input instead of giving in number if it was entered in string. What would happen? Let us see

```
  File "C:\Users\rooman\Anaconda3\lib\site-packages
\spyder_kernels\customize\spydercustomize.py", line 110,
in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

  File "C:/Users/rooman/OneDrive/Desktop/python/test1.py",
line 3, in <module>
    p=int(input('Enter your principal amount'))

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

It has definitely abruptly terminated and the lines below have not executed. More importantly the connection is not been closed. This may result in mischievous problem.

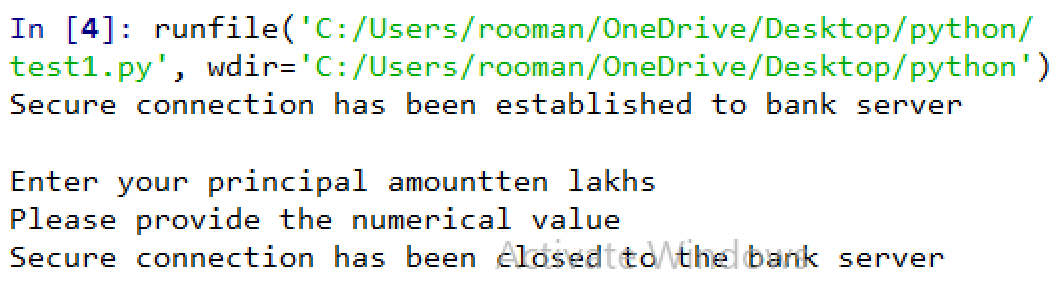
Let us see how can we handle these exceptions and can achieve complete execution of code

- Look for the statement which might arise in logical error. Especially if the input is taken from the users.
- Put those set of instructions with a special block namely "try" and "except" as shown below



```
print('Secure connection has been established to bank server')

try:
    p=int(input('Enter your principal amount'))
    t=int(input('Enter the duration'))
    r=10
    si=(p*t*r)/100
    print('Simple interest =',si)
except:
    print('Please provide the numerical value')

print('Secure connection has been closed to the bank server')
```

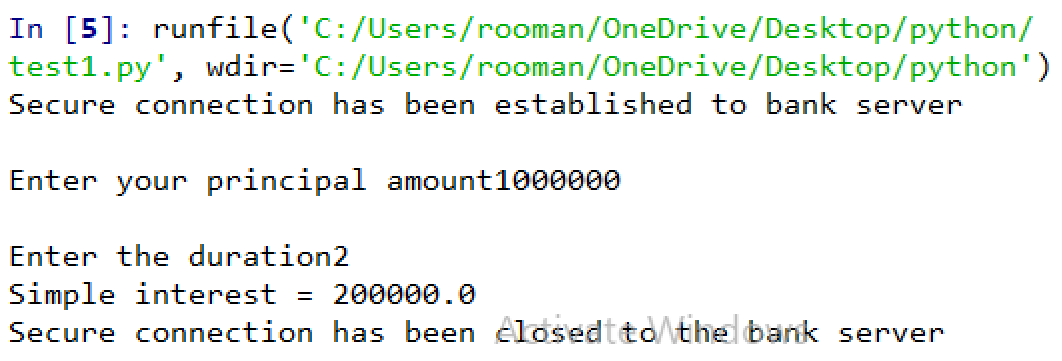
Output:



```
In [4]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test1.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
Secure connection has been established to bank server

Enter your principal amountten lakhs
Please provide the numerical value
Secure connection has been closed to the bank server
```

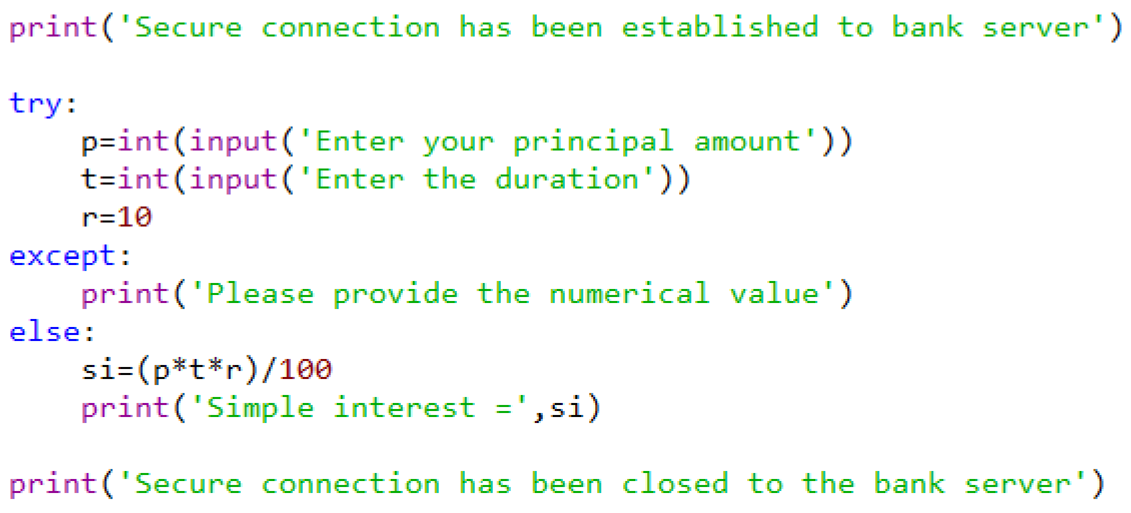
Even though we did not get the expected output, the code executed completely and the connection was closed. Along with it the error message made sense as to why we did not get the expected output.

Note: except block executes only when exception occurs in the try block and an exception object is generated. As shown in below case



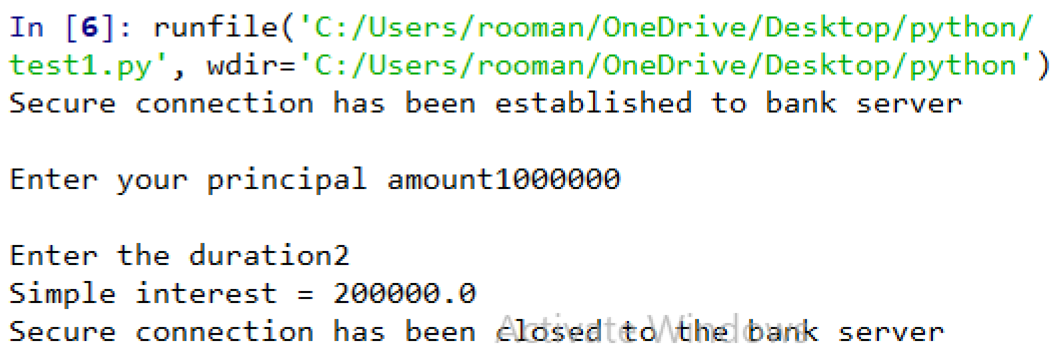
```
In [5]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test1.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
Secure connection has been established to bank server

Enter your principal amount1000000

Enter the duration2
Simple interest = 200000.0
Secure connection has been closed to the bank server
```

There is another block which can be added namely "else" block, under which those statements have to be present which must execute if no exceptions occur in "try" block. As shown below



```
print('Secure connection has been established to bank server')

try:
    p=int(input('Enter your principal amount'))
    t=int(input('Enter the duration'))
    r=10
except:
    print('Please provide the numerical value')
else:
    si=(p*t*r)/100
    print('Simple interest =',si)

print('Secure connection has been closed to the bank server')
```

Output:



```
In [6]: runfile('C:/Users/rooman/OneDrive/Desktop/python/
test1.py', wdir='C:/Users/rooman/OneDrive/Desktop/python')
Secure connection has been established to bank server

Enter your principal amount1000000

Enter the duration2
Simple interest = 200000.0
Secure connection has been closed to the bank server
```