Object Oriented Programming

BE(CSE) II-Semester

Prepared by
S. Durga Devi,
Assistant Professor,
CSE,CBIT

Unit-V

Error and Exception Handling:

- Introduction to errors and exceptions
- Handling Exceptions

GUI Programming

- Simple GUI Programming with tkinter package
- Sample Graphics using Turtle
- Plotting Graphs in Python.

Introduction to Exception Handling

- If you consider any programming language has two types of errors.

1. Syntax error
2. Runtime error

1. Syntax error:

Errors due to invalid syntax.

Ex:

print ‘hai’ # gives syntax error **missing parenthesis in call to print**

- Syntax errors are fixed by programmer, once fixed program can be executed normally.

2. Runtime error: (exceptions)

- Errors during Runtime of the program.

Ex: print(10/0) # ZeroDivisionError: division by zero

print(10/x) # NameError: name 'x' is not defined.

➢ Exception handling is applicable for the handling the run time errors only not for the syntax errors.

Introduction to Exception Handling

What is an Exception?

Exception is a run time error.

An unwanted, unexpected event that disturbs normal flow of the program is called Exception.

ex- our program requirement is read the data from a remote file located in Landon. At runtime if Landon file is not available program is terminated abnormally

exception is that FileNotFoundError.

Is it recommended to handle exception?

- it is highly recommended to handle exceptions under the main objective of Exception Handling.

What is an Exception Handling?

Graceful termination of the program.

defining an alternative way to continue rest of the program execution normally.

- If any thing went wrong we should not miss or lose anything.
- Concept of exception handling is to provide alternative way to continue flow of execution normally without losing anything.
- try:

```
    try:
        read data from a file located at Landon
    except:
            use local file and continue rest of the program normally
```

If the Landon file not available our program should not be terminated abnormally.

we have to provide some local file to continue rest of the program normally. This is nothing but Exception Handling.

- Exception handling does not fix the run time error. It provides the alternative way to continue the execution of rest of the program.

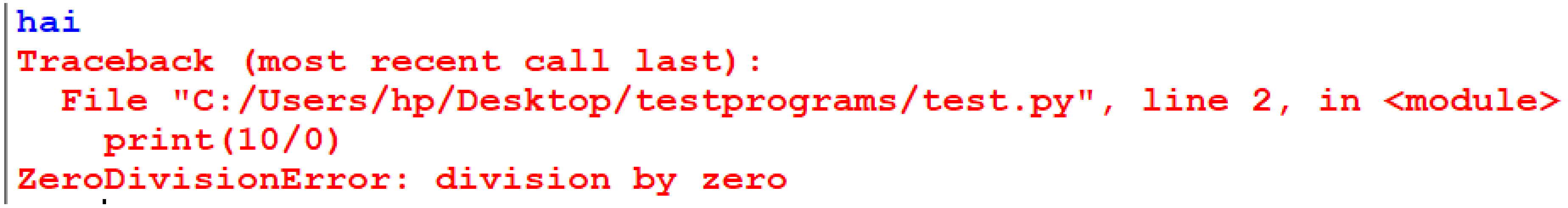
Benefits of exception handling

- Exception handling is used to continue the flow of execution without terminating the program abnormally when run time error is encounter.
- Provides alternative way to continue the program execution normally(grace full termination of program)
- Separates the error code and regular code using try and except blocks.

Default exception handling in python

- Every exception in a python is an object. Every exception has a corresponding class.
- Example:

```
print(‘hai’)
              print(10/0)
              print(‘hello cse1’)
```



```
hai
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/testprograms/test.py", line 2, in <module>
    print(10/0)
ZeroDivisionError: division by zero
```

- When the exception raised in python, python virtual machine(PVM) creates corresponding exception object and checks for the exception handling code in the program. If no handling code in the python, python interpreter terminates the program abnormally and gives the exception report to the program. Then rest of the program will not be executed.

- Your second semester started in January right , your classes are going smoothly without any disturbances. You appeared mid-1 exam also.
- Suddenly covid-19 came to picture which disturbed not only your classes. It has been shaking entire world. To save the people from covid-19 our pm and cm announced lockdown for three weeks.
- all students felt very happy no college, no classes, no travelling nothing. You all enjoyed for week.
- Due to lock down we are unable to run the classes. So covid is an exception because of it our college is not running so no classes.
- What is the solution to run the classes – conduct online classes.
- we are handling this lock down period by conducting online classes.
- Conducting online classes with out interrupting the classes is the exception handling.

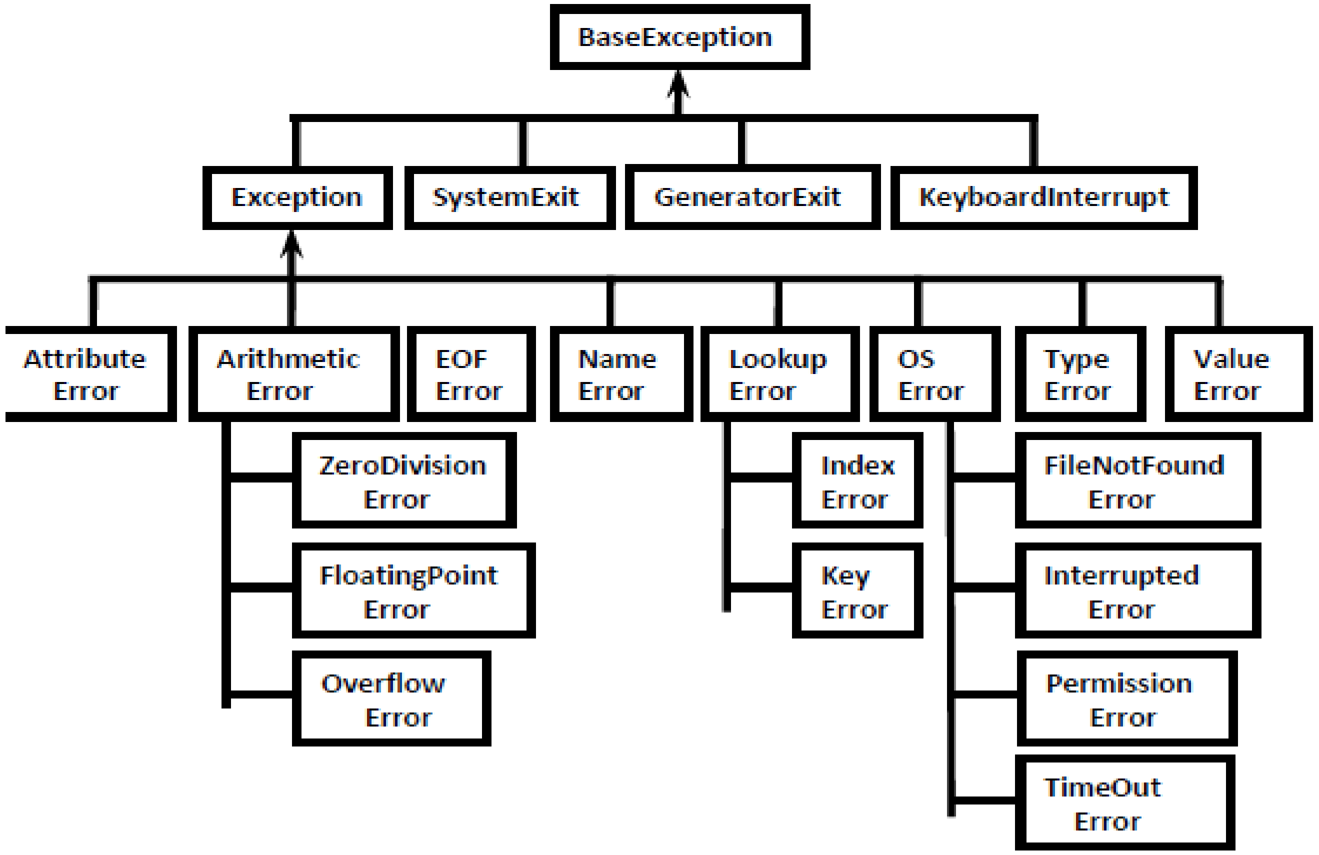
Exception is a base class for all exceptions

Built in Exception in python

- AttributeError- attribute assignment or reference fails
- ImportError- module not found
- NameError- when a variable not found in local or global space.
- ZeroDivisionError- when a number divides with zero.
- ValueError: when a function receives a correct type but inappropriate value.

```
import math
math.sqrt(-10)
```

Exception hierarchy



https://www.youtube.com/watch?v=UtT_LaEBimU&list=PLd3UqWTnYXOlBZgb711WywNSHPsJIedtU&index=3

Handling exceptions(try,except,finally)

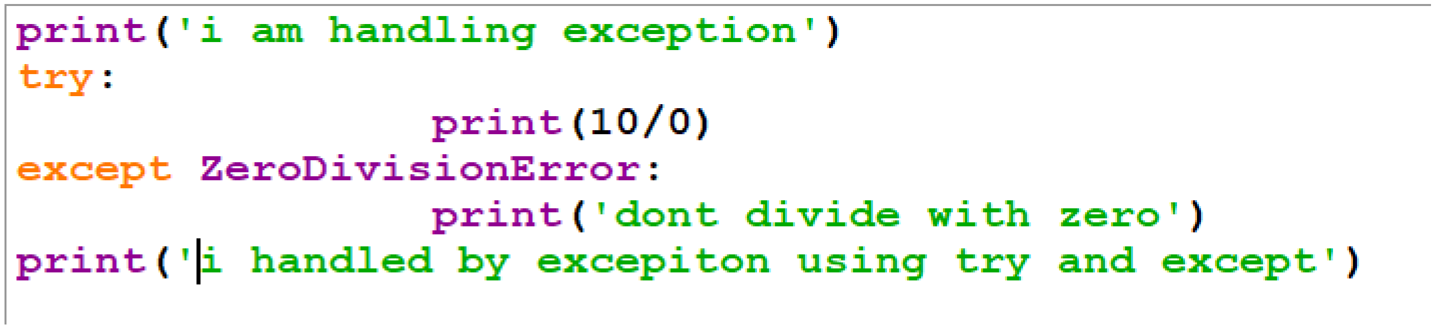
- try ,except and finally keywords are used to handle exceptions.
- Syntax:

```
 try:
          riskycode
 except  Exceptionclass:
           handling code
finally:
          cleanup code
```

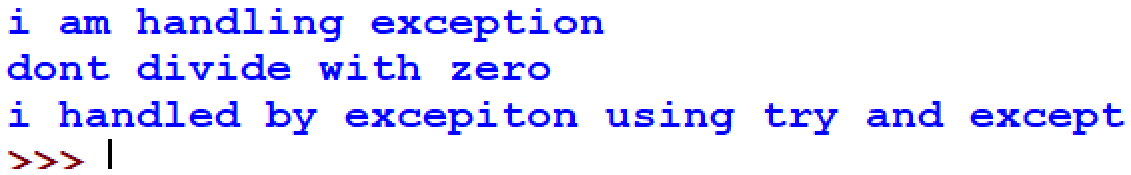
- Where, risky code means- code that raises the exception
  handling code – alternative way.

Note: using finally block is optional.

try and except example



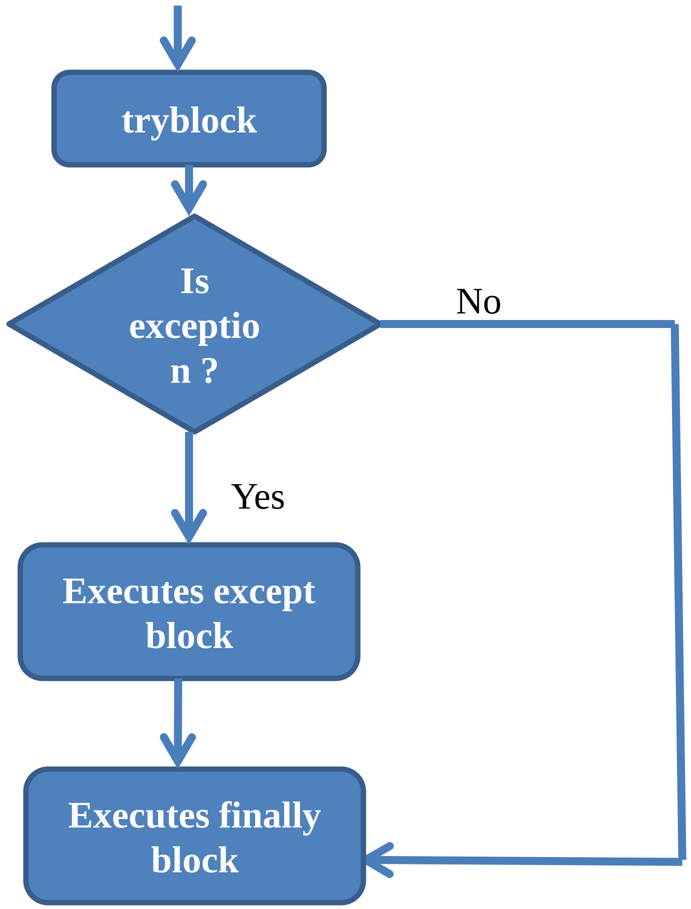
```python
print('i am handling exception')
try:
                print(10/0)
except ZeroDivisionError:
                print('dont divide with zero')
print('i handled by excepiton using try and except')
```



```
i am handling exception
dont divide with zero
i handled by excepiton using try and except
>>> 
```

Program terminated normally though exception raised.

Control flow in try and except



S. Durga Devi ,CSE,CBIT

- In which order statements are executed if exception rised?

```
try:
   statement1;
   statement2;
    statement3;
except:
   statement4;

 statement5;
```

Case1:- no exception
1,2,3&5 are executed

Case2:- if exception rised at statement2
1,4,5 are executed.
once it comes out of try block control never goes back. Rest of the try block never executed
so write only risky code inside of the try block.

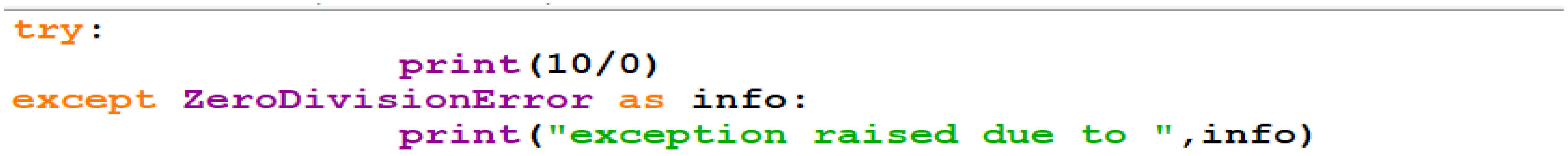
Case 3:- if exception rised at statement4
exceptions can be rised in exceptand finally blocks also sometimes
- causes abnormal termination.

Case4:- if exception rised in try block and corresponding except block is not present
- causes abnormal termination.

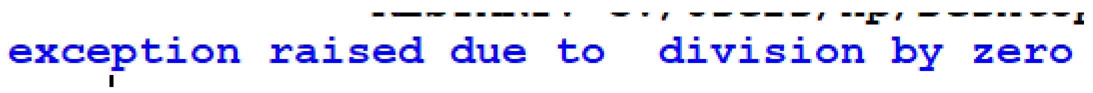
Note

within the try block if any where an exception rised then the rest of try block wont be executed even though we handled that exception.
hence within the try block we have to take only risky code and length of the try block should be as less as possible.

How to print the exception information ?



```
try:
                    print(10/0)
except ZeroDivisionError as info:
                    print("exception raised due to ",info)
```



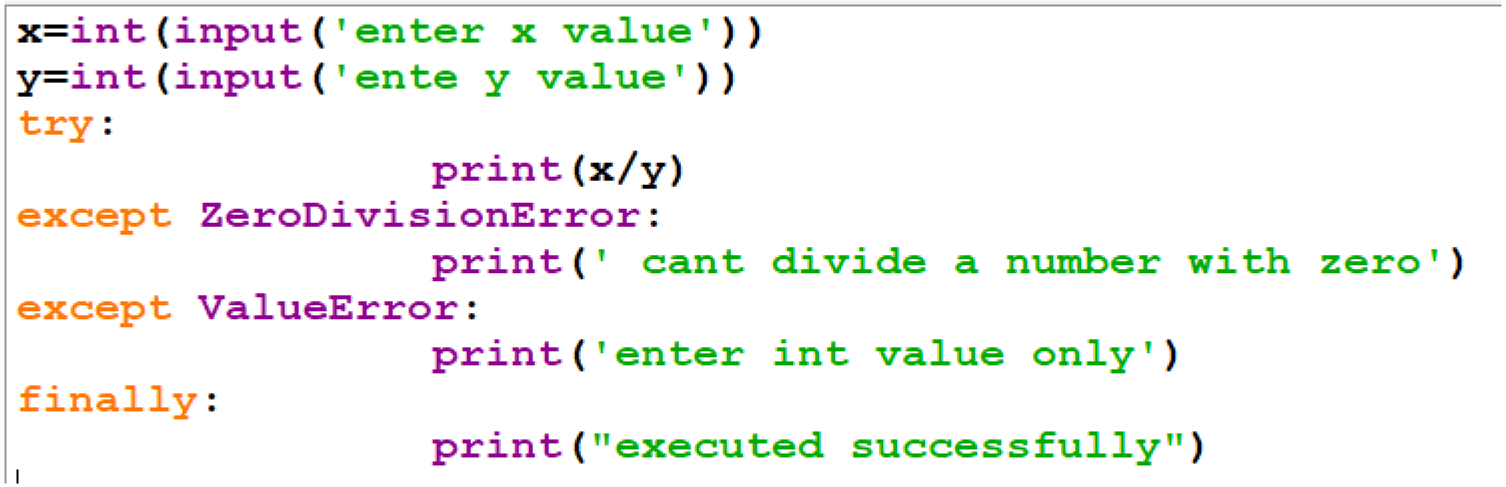
```
 exception raised due to  division by zero
```

S. Durga Devi ,CSE,CBIT

try with multiple except blocks

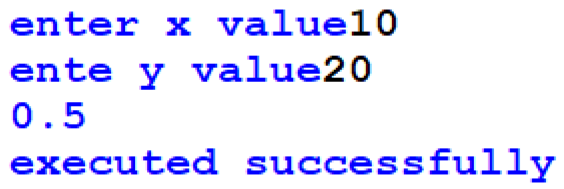
- If programmer does not know what type exception will rise in the code then go for the multiple except blocks.
- For every exception a separate except block is to be defined.
- When exception is thrown, each except block is inspected in order, and the first one which matches with the type of exception that except block will be executed.
- When multiple except blocks used in a program you must remember in which order exception types should be specified for except block.
- Exception subclasses should come before its super class exceptions why because except statement which has super class will handle all its sub classes exceptions .
- It never give chance to handle for its sub class

Example on multiple except blocks



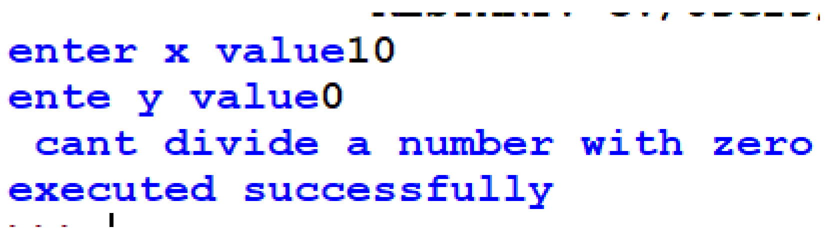
```
x=int(input('enter x value'))
y=int(input('ente y value'))
try:
                print(x/y)
except ZeroDivisionError:
                print(' cant divide a number with zero')
except ValueError:
                print('enter int value only')
finally:
                print("executed successfully")
```

testcase1



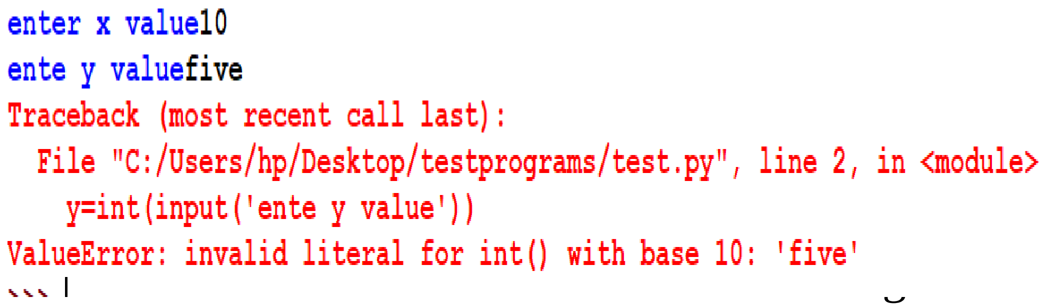
```
enter x value10
ente y value20
0.5
executed successfully
```

testcase2



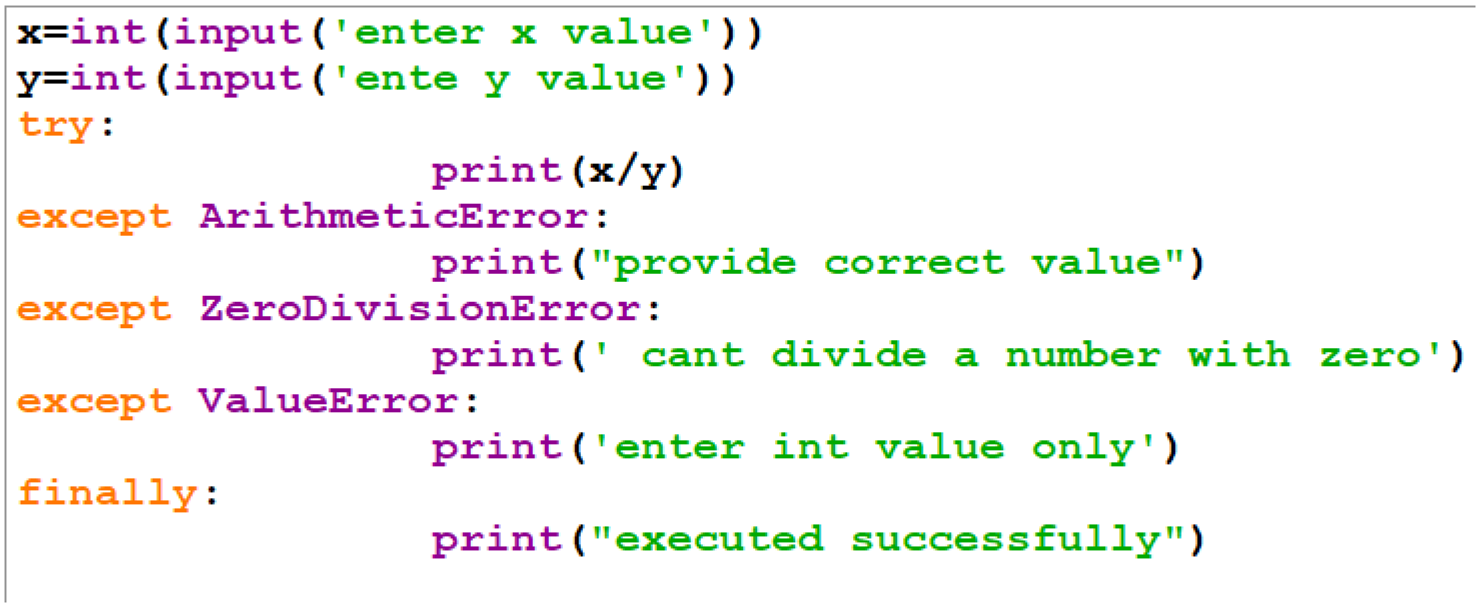
```
enter x value10
ente y value0
 cant divide a number with zero
executed successfully
```

testcase3

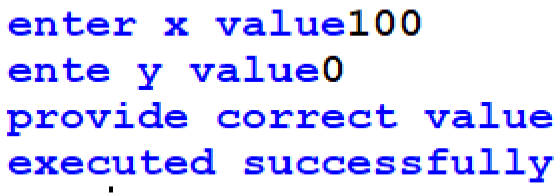


```
enter x value10
ente y valuefive
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/testprograms/test.py", line 2, in <module>
    y=int(input('ente y value'))
ValueError: invalid literal for int() with base 10: 'five'
```

- Exception subclasses should come before its super class exceptions why because except statement which has super class will handle all its sub classes exceptions .
- It never give chance to handle for its sub class



```
x=int(input('enter x value'))
y=int(input('ente y value'))
try:
                print(x/y)
except ArithmeticError:
                print("provide correct value")
except ZeroDivisionError:
                print(' cant divide a number with zero')
except ValueError:
                print('enter int value only')
finally:
                print("executed successfully")
```



```
enter x value100
ente y value0
provide correct value
executed successfully
```

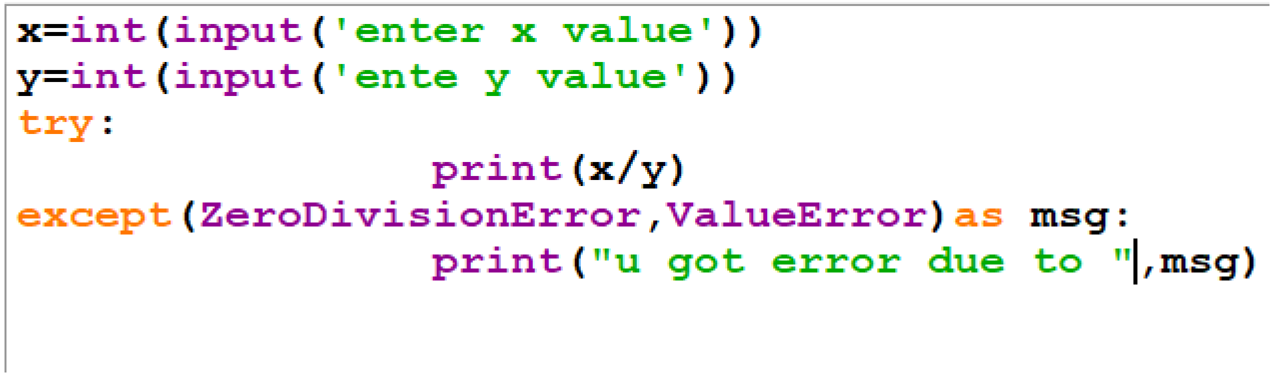
Single except block handles multiple exceptions

- A single except block can handle multiple exceptions using following syntax.
- Syntax:

except(exception1,exception2,……) or

except(exception1,exception2,…..) as info

Note: parenthesis is must and will consider internally as tuple



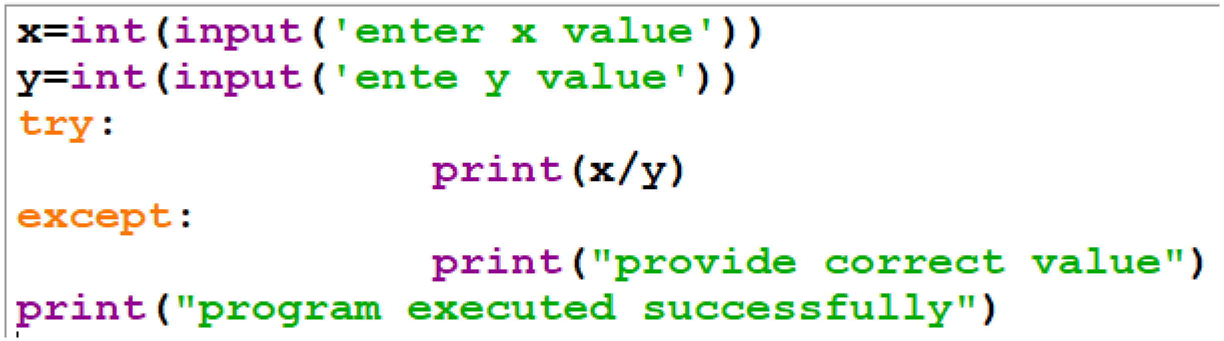
```
x=int(input('enter x value'))
y=int(input('ente y value'))
try:
                print(x/y)
except(ZeroDivisionError,ValueError)as msg:
                print("u got error due to ",msg)
```

Default except block

- To handle any type of error we can use default except block.
- Syntax:

except:

statement



```
x=int(input('enter x value'))
y=int(input('ente y value'))
try:
                print(x/y)
except:
                print("provide correct value")
print("program executed successfully")
```

Note: If try with multiple except blocks available then default except block should be last,otherwise we will get SyntaxError.

How many ways we can use except block

1. except ValueError:
2. except ValueError as info:
3. except(ValueError,TypeError):
4. except(ValueError,TypeError) as info
5. except:

-

finally keyword

- **finally keyword.**

finally is a block always associated with try and except to maintain clean up code.

- What is specialty of finally?

finally block is always executed irrespective of whether exception is rised or not rised and whether handled or not handled.

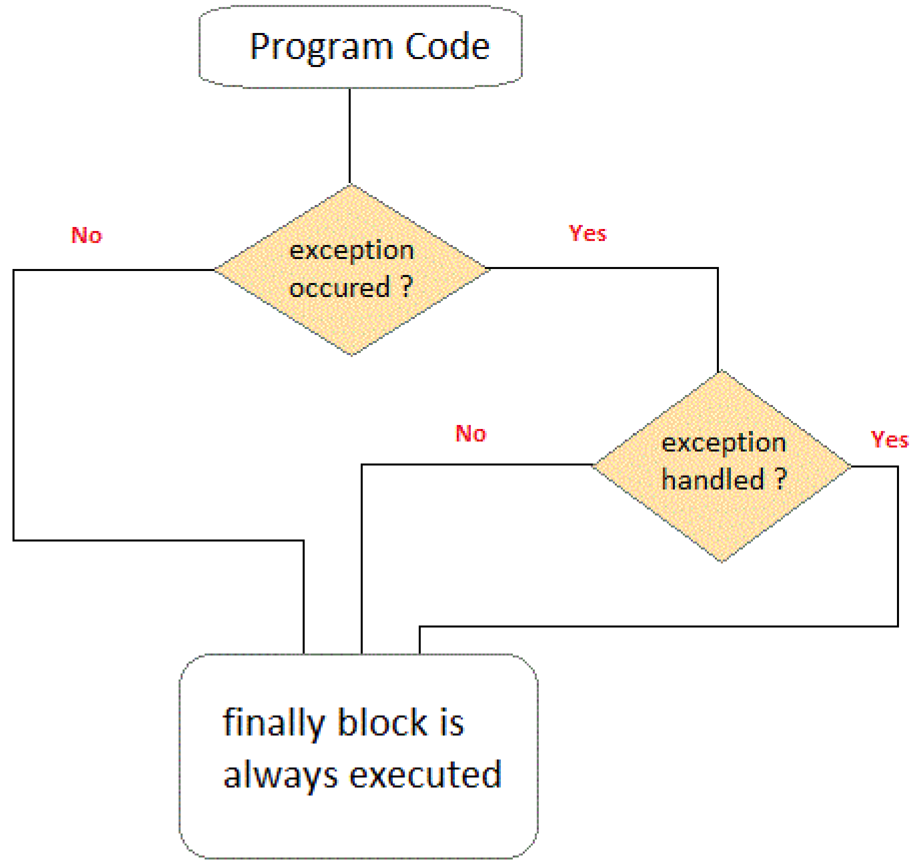
- whatever resources are allocated in the try block can be deallocated by finally block.

General form of finally.

```
try:
        risky code


except:
        handling code
finally:
      clean up code
```

finally block



- Case study where finally block is more useful
- Suppose open a file in read mode and read first line of a file and display it.
- 1. f=open(‘test.py’,’r’)

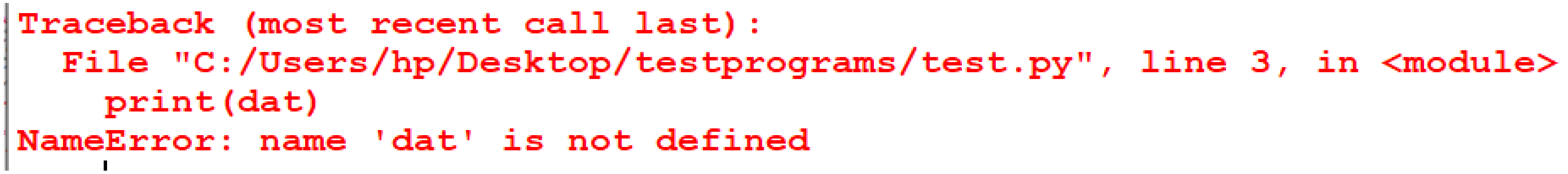
  2. data=f.readline()

  3. print(dat)# NameError

  4. f.close()

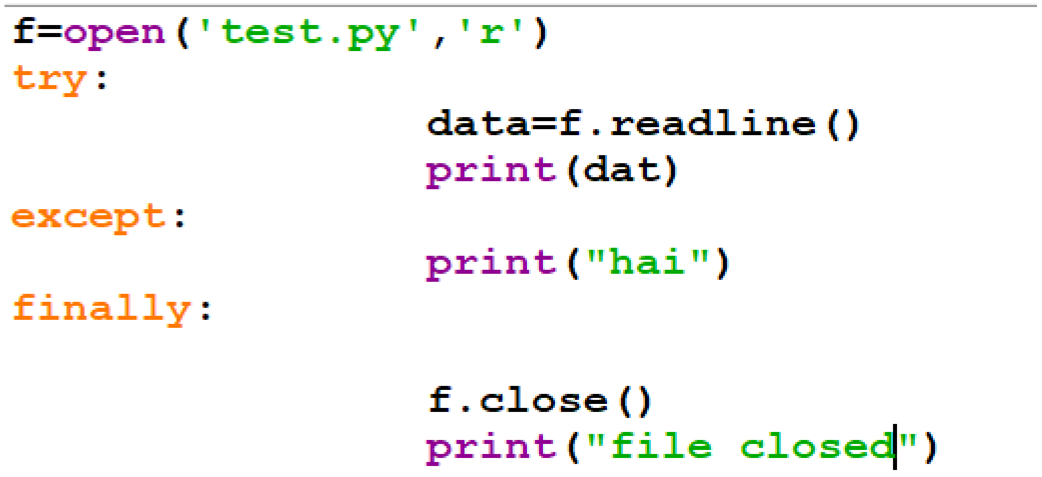
  in above code exception at line no 3. file unable to close. Then use finally block.

output



```
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/testprograms/test.py", line 3, in <module>
    print(dat)
NameError: name 'dat' is not defined
```

finally example



```
f=open('test.py','r')
try:
                data=f.readline()
                print(dat)
except:
                print("hai")
finally:

                f.close()
                print("file closed")
```

Output

```
hai
file closed
```

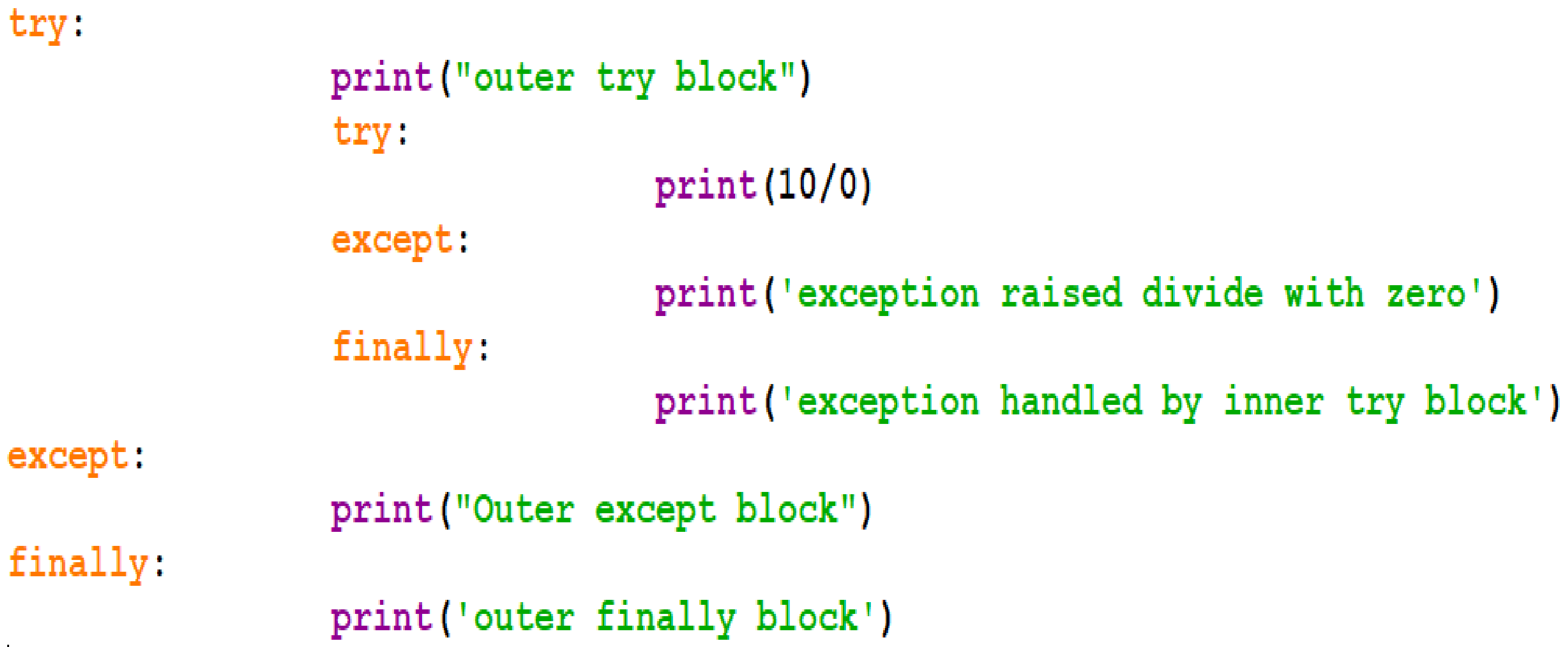
Nested try-except-finally blocks:

- We can define try-except-finally blocks inside the try, except and finally blocks i.e nested try-except-finally blocks.

```
try:# outer try block
      st1
      st2
      try: #inner try block
              st3
      except:
                st4
       finally:
                st5
except:
                st6
 finally:
                st7
 st8
```

- when exception rised in inner try block, inner except block will handle exception. If inner except block unable to handle, then outer except block will handle.

Example on nested try-except-finally



```
try:
    print("outer try block")
    try:
        print(10/0)
    except:
        print('exception raised divide with zero')
    finally:
        print('exception handled by inner try block')
except:
    print("Outer except block")
finally:
    print('outer finally block')
```

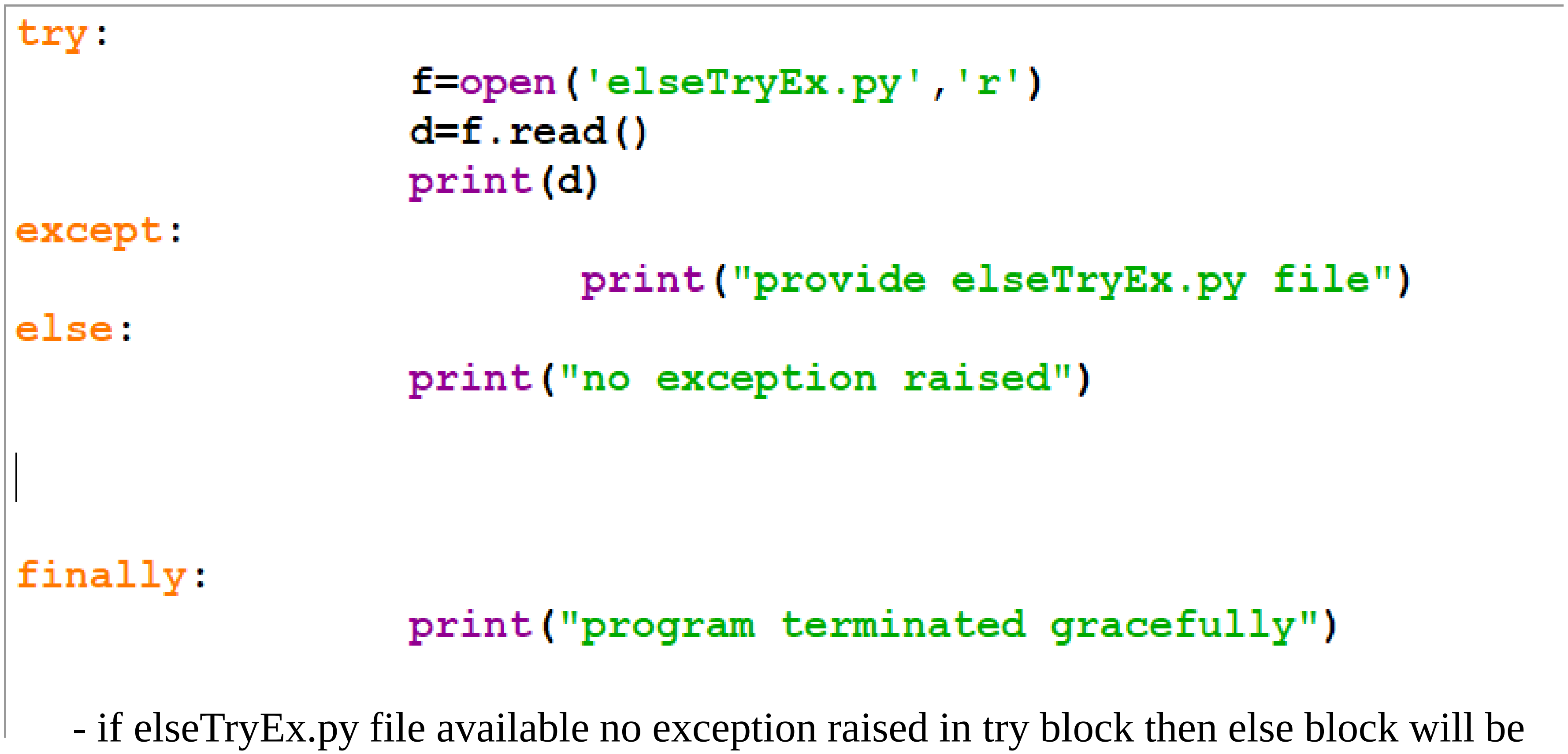
else clause with try-except-finally

- Along with try-except-finally blocks we can also use **else** clause.
- else block will be executed when try block does not rise the exception.

Syntax:

```
 try:
             errorcode
except:
             executed when exception rised
else:
             executed when no exception
 finally:
             executed irrelevant of the exception
```

Example on else clause



```
try:
                f=open('elseTryEx.py','r')
                d=f.read()
                print(d)
except:
                       print("provide elseTryEx.py file")
else:
                print("no exception raised")


finally:
                print("program terminated gracefully")
```

- if elseTryEx.py file available no exception raised in try block then else block will be executed.
- if exception raised else block will not be executed.

Types of exceptions

- There are two types of exceptions in python

1. predefined\built in exceptions
2. User defined exceptions\customized exceptions.

1. Predefined exceptions:

- when the exception raised in python program, python virtual machine creates exception object and stops the program execution and gives the information to the programmer.

Ex: ZeroDivisionError, ValueError, TypeError etc.

2. user defined exceptions

- user or programmer can raise his own exceptions explicitly to deal with common problems such as

a) attempting to enter invalid PIN in ATM transactions.-InvalidPinError

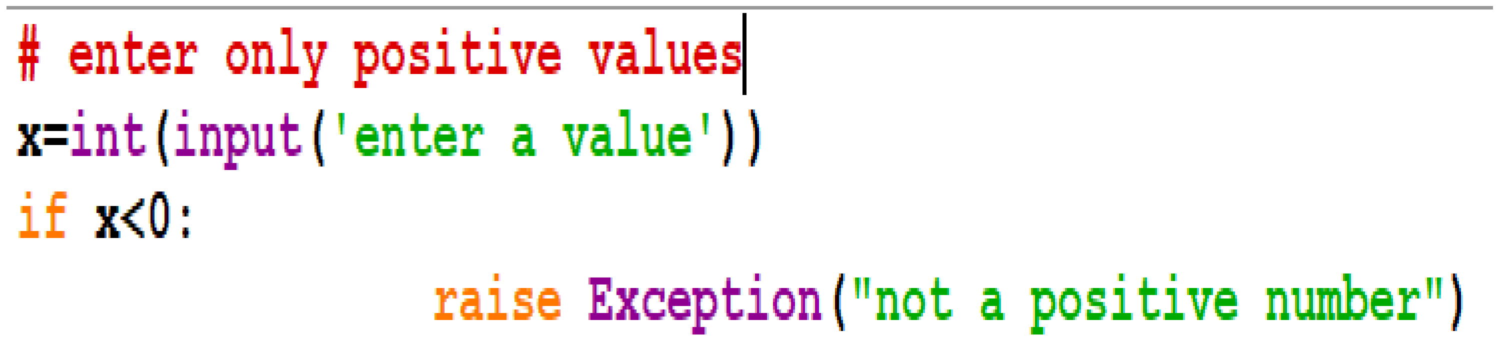
b) attempt to enter negative salary for an employee- NegativeSalaryError

c) attempt to enter wrong username and password- WrongDetailsError

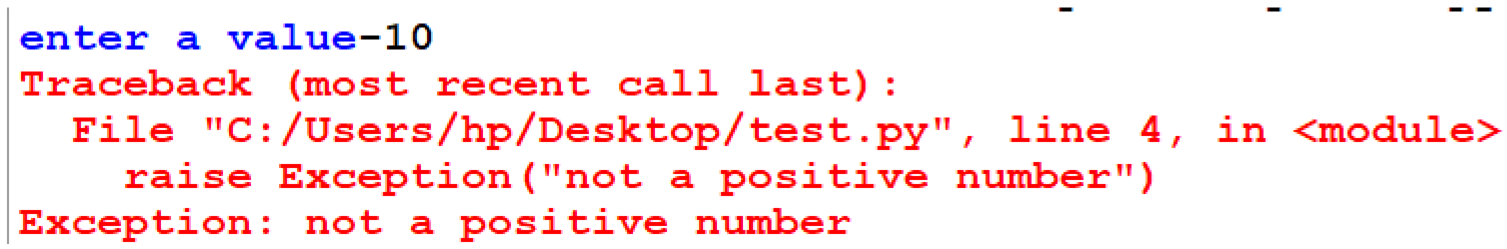
d) attempt to withdraw more amount than existing.-InsufficientFundsError

raise exceptions by user

- ‘ raise’ statement is used to raise the exceptions forcefully by the user rather than python virtual machine.
- Example



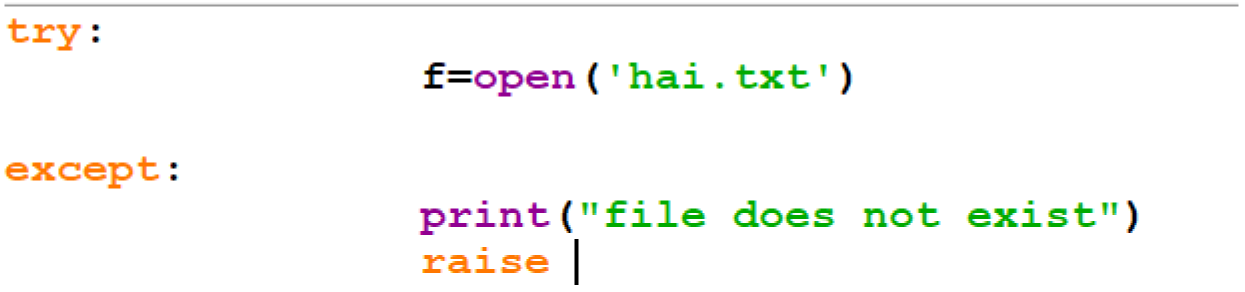
```
# enter only positive values
x=int(input('enter a value'))
if x<0:
            raise Exception("not a positive number")
```



```
enter a value-10
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/test.py", line 4, in <module>
    raise Exception("not a positive number")
Exception: not a positive number
```

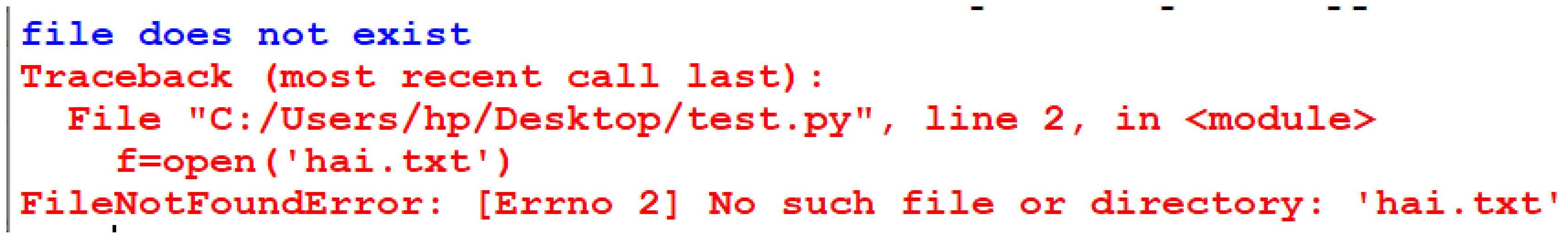
Re-raise the exception

- We can also reraise the excpetion. This will be used when exception raised you dont intend to handle it.



```
try:
                    f=open('hai.txt')

except:
                    print("file does not exist")
                    raise
```



```
file does not exist
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/test.py", line 2, in <module>
    f=open('hai.txt')
FileNotFoundError: [Errno 2] No such file or directory: 'hai.txt'
```

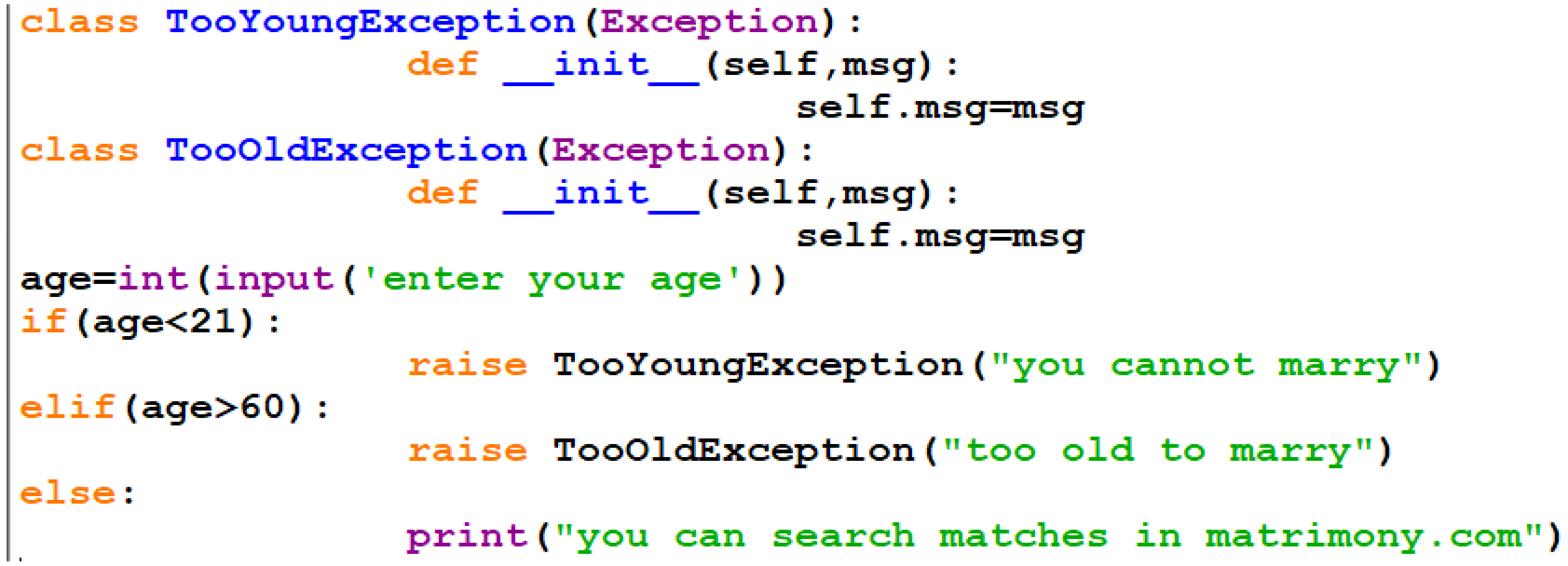
How to define and raise customized exceptions

- Exception is super class for all the exceptions in python.
- User defined exceptions are defined by create a class that extends Exception class.
- Raise the exception using keyword 'raise'.
- raise is used to forcefully raise the exception by the user.

- Syntax:

```
class classname(Exception):
        def __init__(self,msg):
                self.msg=msg
```

Example on customized exceptions



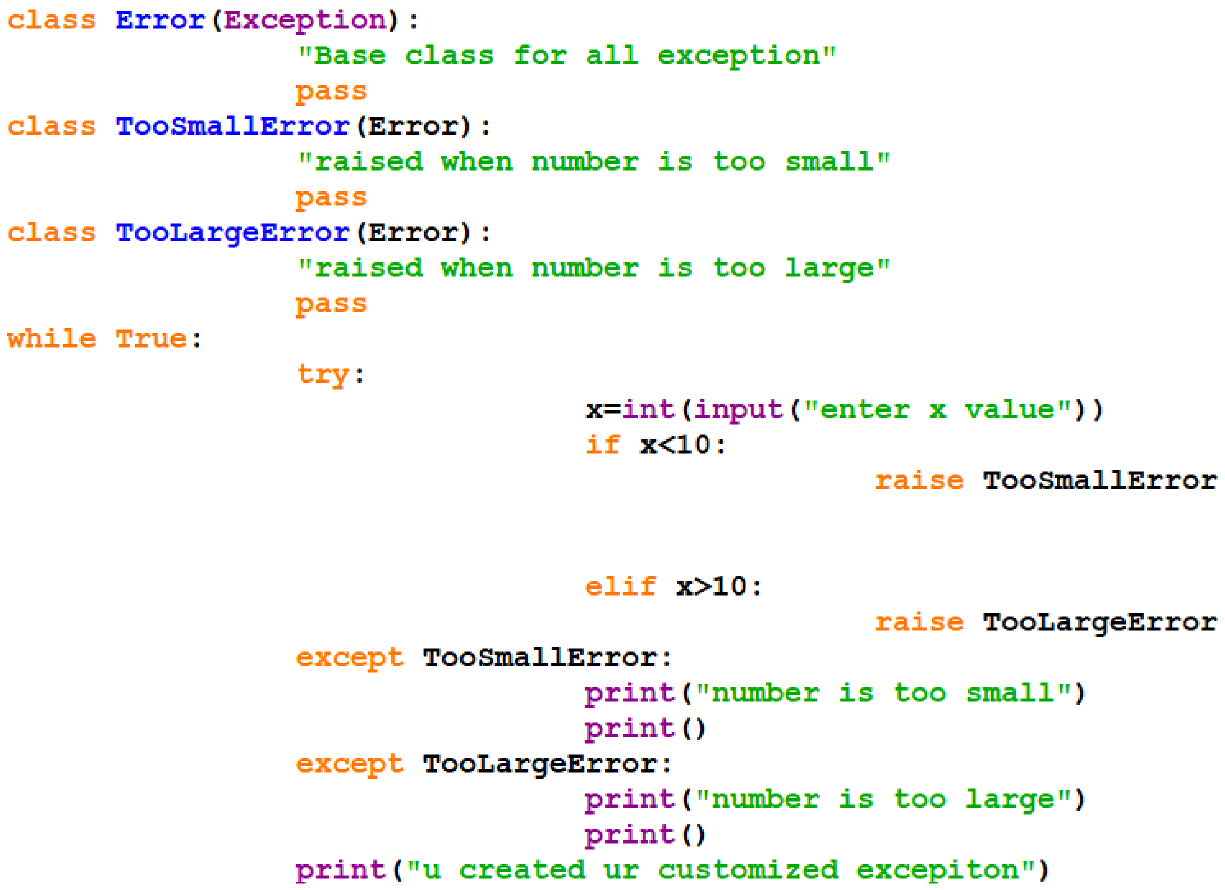
```
class TooYoungException(Exception):
                def __init__(self,msg):
                                self.msg=msg
class TooOldException(Exception):
                def __init__(self,msg):
                                self.msg=msg
age=int(input('enter your age'))
if(age<21):
                raise TooYoungException("you cannot marry")
elif(age>60):
                raise TooOldException("too old to marry")
else:
                print("you can search matches in matrimony.com")
```

Customized exception handling

```
class TooYoungError(Exception):
    pass
class TooOldError(Exception):
    pass
try:
    age=int(input("Enter your age"))
    if(age<21):
        raise TooYoungError
    elif(age>60):
        raise TooOldError
    else:
        print("you can marry now")
except TooYoungError:
    print("you are too young to marry")
except TooOldError:
    print("you are too old to marry")
```

User defined exceptions

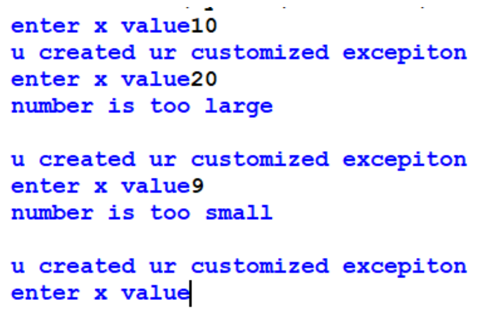
- Every user defined class must extends base class Exception
- Exception is base class for all the exceptions



```
class Error(Exception):
        "Base class for all exception"
        pass
class TooSmallError(Error):
        "raised when number is too small"
        pass
class TooLargeError(Error):
        "raised when number is too large"
        pass
while True:
        try:
                x=int(input("enter x value"))
                if x<10:
                        raise TooSmallError


                elif x>10:
                        raise TooLargeError
        except TooSmallError:
                print("number is too small")
                print()
        except TooLargeError:
                print("number is too large")
                print()
        print("u created ur customized excepiton")
```

output



```
enter x value10
u created ur customized excepiton
enter x value20
number is too large

u created ur customized excepiton
enter x value9
number is too small

u created ur customized excepiton
enter x value
```

Durga Devi, CBIT, CSE

Traceback in python

- Traceback maintains the function calls report in the program
- Also called stack trace.
- When exception raised in the program python virtual machine prints traceback to help you to know what went wrong in your program.

```
class A:
                def m1(self):
                                print(10/0)
class B(A):
                def m2(self):
                                self.m1()

x=B()
x.m2()
```

```
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/test.py", line 9, in <module>
    x.m2()
  File "C:/Users/hp/Desktop/test.py", line 6, in m2
    self.m1()
  File "C:/Users/hp/Desktop/test.py", line 3, in m1
    print(10/0)
ZeroDivisionError: division by zero
```

Assertions in python

- Assertions are used for the debugging purposes.
- Debugging is process of finding and fixing bugs.
- Two types of assertions

1. Simple version
2. Augmented version

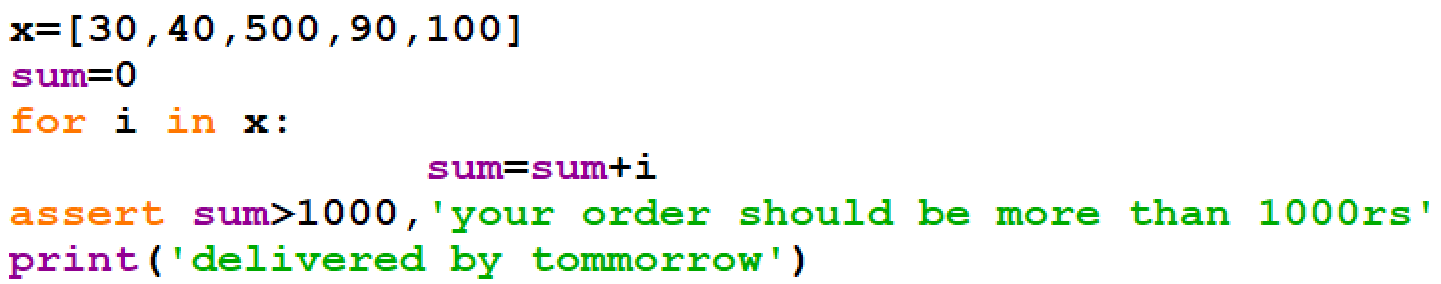
   1. Simple version

      assert conditional_expression

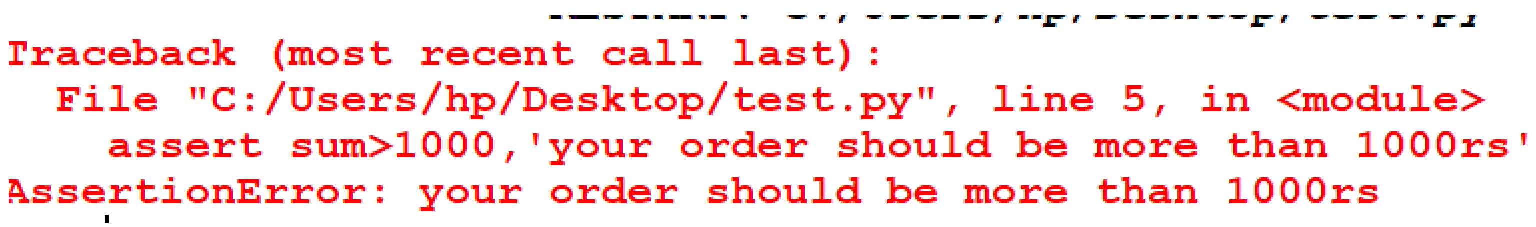
   2. augmented version

      assert conditinal_ expression, msg

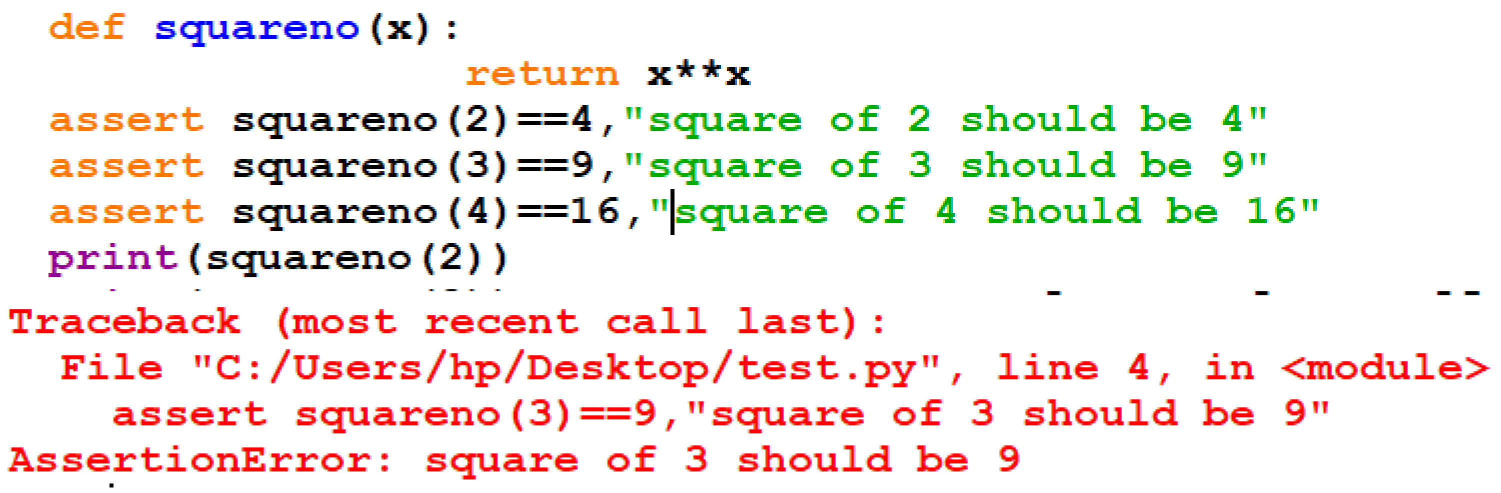
- conditional_expression will be evaluated and if it is true then the program will be continued.
- If it is false then the program will be terminated by raising AssertionError.
- By seeing AssertionError, programmer can analyse the code and can fix the problem



```
x=[30,40,500,90,100]
sum=0
for i in x:
                sum=sum+i
assert sum>1000,'your order should be more than 1000rs'
print('delivered by tommorrow')
```

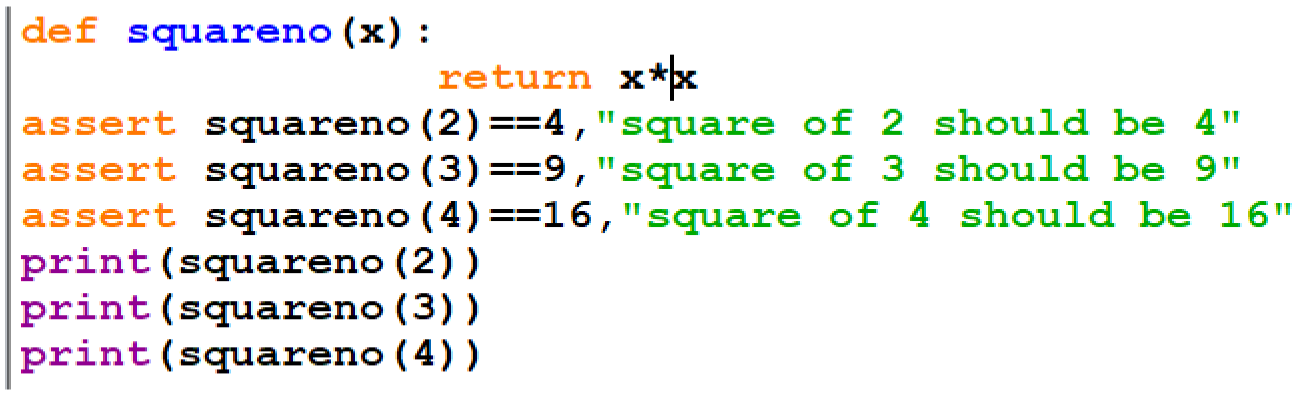


```
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/test.py", line 5, in <module>
    assert sum>1000,'your order should be more than 1000rs'
AssertionError: your order should be more than 1000rs
```



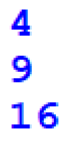
```
def squareno(x):
                return x**x
assert squareno(2)==4,"square of 2 should be 4"
assert squareno(3)==9,"square of 3 should be 9"
assert squareno(4)==16,"square of 4 should be 16"
print(squareno(2))
```

```
Traceback (most recent call last):
  File "C:/Users/hp/Desktop/test.py", line 4, in <module>
    assert squareno(3)==9,"square of 3 should be 9"
AssertionError: square of 3 should be 9
```



```python
def squareno(x):
                return x*x
assert squareno(2)==4,"square of 2 should be 4"
assert squareno(3)==9,"square of 3 should be 9"
assert squareno(4)==16,"square of 4 should be 16"
print(squareno(2))
print(squareno(3))
print(squareno(4))
```

output



```
4
9
16
```