

Exception Handling

BBM 101 - Introduction to Programming I

Hacettepe University

Fall 2015

Fuat Akal, Aykut Erdem, Erkut Erdem, Vahid Garousi

What is an Exception?

- An exception is an abnormal condition that arises in a code sequence at runtime. For instance:
 - Dividing a number by zero
 - Accessing an element that is out of bounds of an array
 - Attempting to open a file which does not exist
- When an exceptional condition arises, an object representing that exception is created and thrown in the code that caused the error
- An exception can be caught to handle it or pass it on
- Exceptions can be generated by the run-time system, or they can be manually generated by your code

What is an Exception?

```
1 dividend = 5
2 divisor = 0
3 division = dividend / divisor
4 print "Result = " + str(division)
```

Program crashes on 3rd line!

Traceback (most recent call last):

*File "/Users/akal/Programs/PycharmProjects/untitled/bbm101.py", line 5, in
<module>*

division = dividend / divisor

ZeroDivisionError: integer division or modulo by zero

What is Exception Handling?

- Exception mechanism gives the programmer a chance to do something against an abnormal condition.
- Exception handling is performing an action in response to an exception.
- This action may be:
 - Exiting the program
 - Retrying the action with or without alternative data
 - Displaying an error message and warning user to do something
 -

What is Exception Handling?

```
1     try:  
2         dividend = 5  
3         divisor = 0  
4         division = dividend / divisor  
5         print "Result = " + str(division)  
6     except:  
7         print "Exception occurred and handled!"
```

Your program now prints a nicer message:

Exception occurred and handled!

What Really Happened

- **division = dividend / divisor** statement causes an exception
- Python run-time system throws an exception object that includes data about the exception
- Execution is stopped at the 4th line, and an except block is searched to handle the exception
- Exception is caught by the 6th line and execution continues by the 7th line
- Output of the program is:

Exception occurred and handled!

Keywords of Exception Handling

- There are five keywords in Python to deal with exceptions: **try**, **except**, **else**, **raise** and **finally**.
- **try**: Creates a block to monitor if any exception occurs.
- **except**: Follows the try block and catches any exception which is thrown within it.

Are There Many Exceptions in Python?

- Yes, some of them are...
 - Exception
 - ArithmeticError
 - OverflowError
 - ZeroDivisonError
 - EOFError
 - NameError
 - IOError
 - SyntaxError

Multiple except Statements

- It is possible that more than one exception can be thrown in a code block.
 - We can use multiple **except** clauses
- When an exception is thrown, each **except** statement is inspected in order, and the first one whose type *matches* that of the exception is executed.
 - Type matching means that the exception thrown must be an object of the same class or a sub-class of the declared class in the **except** statement
- After one **except** statement executes, the others are bypassed.

Multiple except Statements

try:

You do your operations here;

except Exception-1:

Execute this block.

except Exception-2:

Execute this block.

except (Exception-3[, Exception-4[,...ExceptionN]]):

If there is any exception from the given exception list,
then execute this block.

Multiple except Statements

```
try:
    f = open('outfile.dat', 'w')
    dividend = 5
    divisor = 0
    division = dividend / divisor
    f.write(str(division))
except IOError:
    print "I can't open the file!"
except ZeroDivisionError:
    print "You can't divide by zero!"
```

You can't divide by zero!

Multiple except Statements

```
try:
    f = open('outfile.dat', 'w')
    dividend = 5
    divisor = 0
    division = dividend / divisor
    f.write(str(division))
except Exception:
    print "Exception occurred and handled!"
except IOError:
    print "I can't open the file!"
except ZeroDivisionError:
    print "You can't divide by zero!"
```

Exception occurred and handled!

Multiple except Statements

try:

```
f = open('outfile.dat', 'w')
dividend = 5
divisor = 0
division = dividend / divisor
f.write(str(division))
```

except:

```
print "Exception occurred and handled!"
```

except IOError:

```
print "I can't open the file!"
```

except ZeroDivisionError:

```
print "You can't divide by zero!"
```

SyntaxError: default 'except:' must be last

except-else Statements

try:

You do your operations here

except:

Execute this block.

else:

If there is no exception then execute this block.

```
try:
    f = open(arg, 'r')
except IOError:
    print 'cannot open', arg
else:
    print arg, 'has', len(f.readlines()), 'lines'
```

finally Statement

- **finally** creates a block of code that will be executed after a **try/except** block has completed and before the code following the **try/except** block
- **finally** block is executed whether or not exception is thrown
- **finally** block is executed whether or not exception is caught
- It is used to guarantee that a code block will be executed in any condition.

finally Statement

You can use it to clean up files, database connections, etc.

try:

You do your operations here

except:

Execute this block.

finally:

This block will definitely be executed.

```
try:
    file = open('out.txt', 'w')
    do something...
finally:
    file.close()
    os.path.remove('out.txt')
```


Nested try Blocks

- When an exception occurs inside a **try** block;
 - If the **try** block does not have a matching except, then the outer **try** statement's except clauses are inspected for a match
 - If a matching except is found, that except block is executed
 - If no matching except exists, execution flow continues to find a matching except by inspecting the outer try statements
 - If a matching except cannot be found at all, the exception will be caught by Python's exception handler.
- Execution flow never returns to the line that exception was thrown. This means, an exception is caught and except block is executed, the flow will continue with the lines following this except block

Let's clarify it on various scenarios

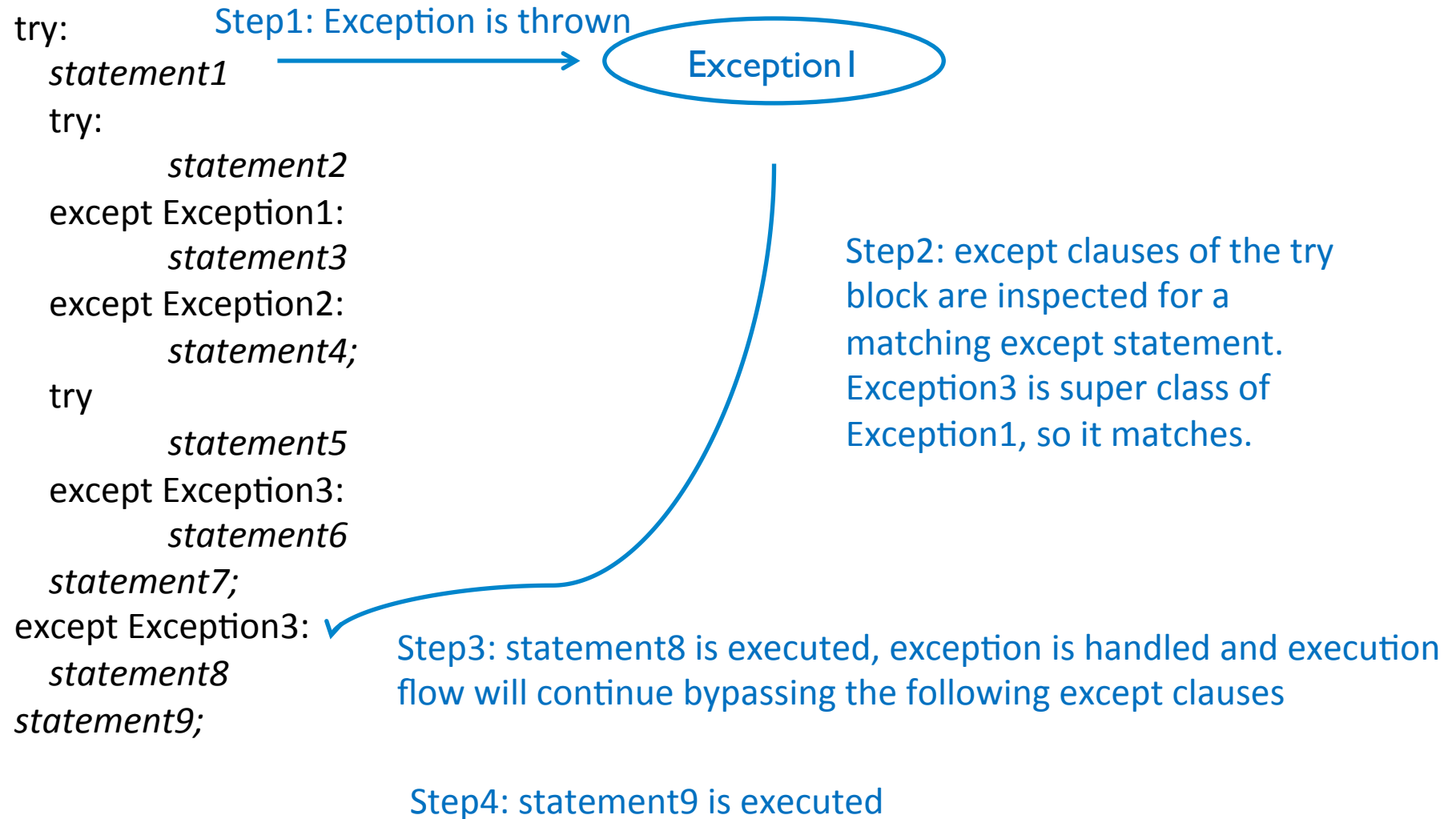
```
try:
    statement1
    try:
        statement2
    except Exception1:
        statement3
    except Exception2:
        statement4;
    try
        statement5
    except Exception3:
        statement6
    statement7;
except Exception3:
    statement8
statement9;
```

Information: Exception1 and Exception2 are subclasses of Exception3

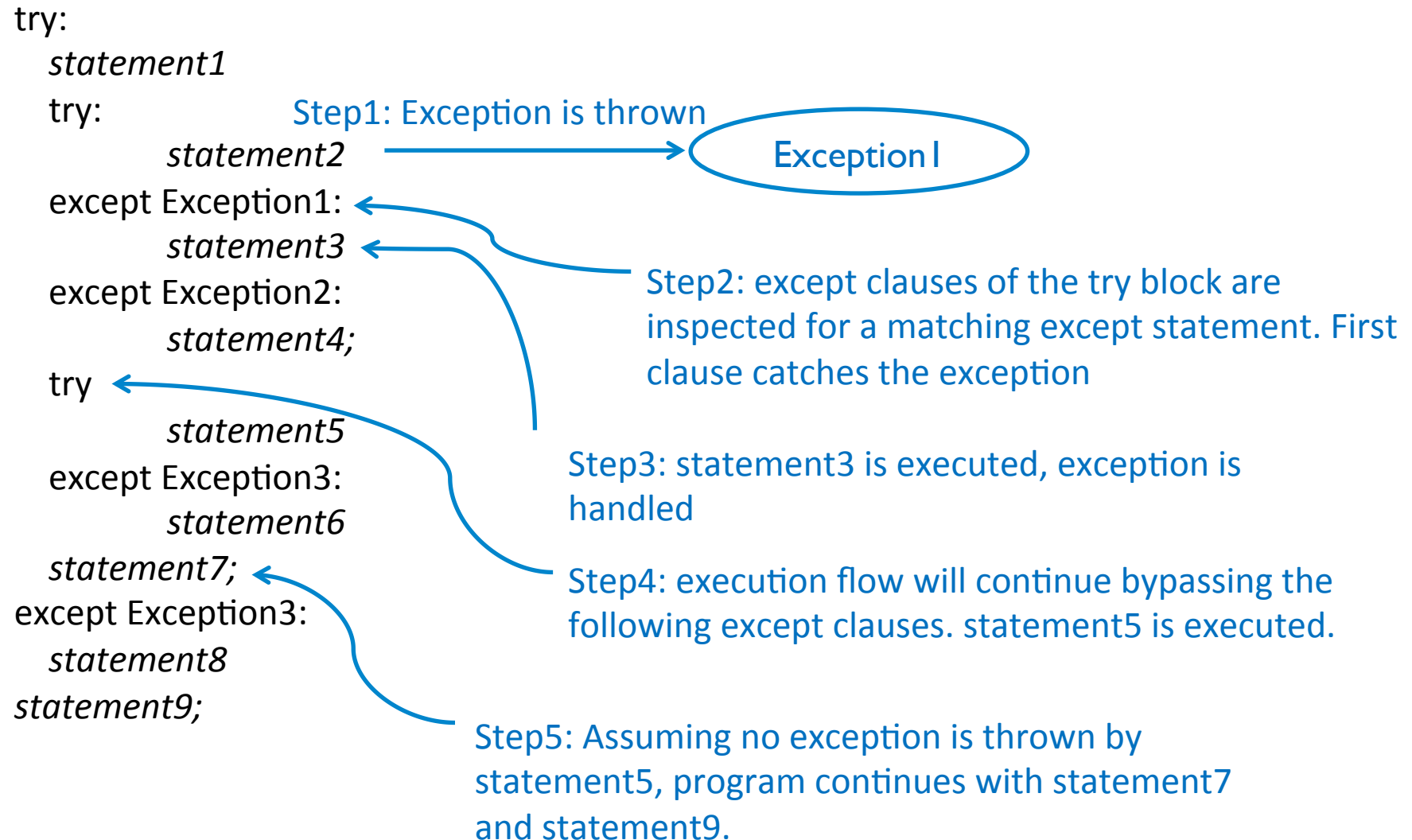
Question: Which statements are executed if

- 1- statement1 throws Exception1
- 2- statement2 throws Exception1
- 3- statement2 throws Exception3
- 4- statement2 throws Exception1 and statement3 throws Exception2

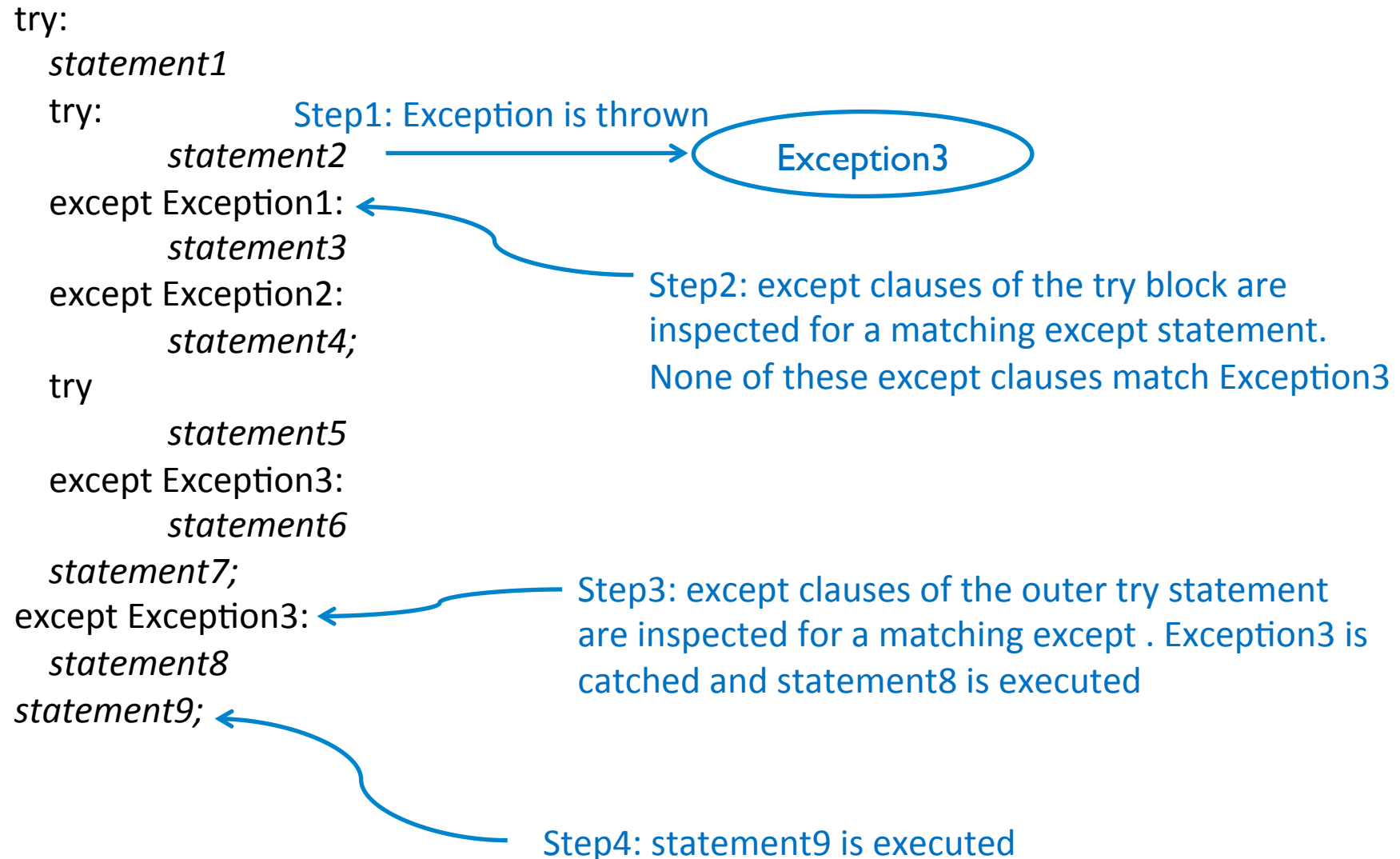
Scenario: statement1 throws Exception1



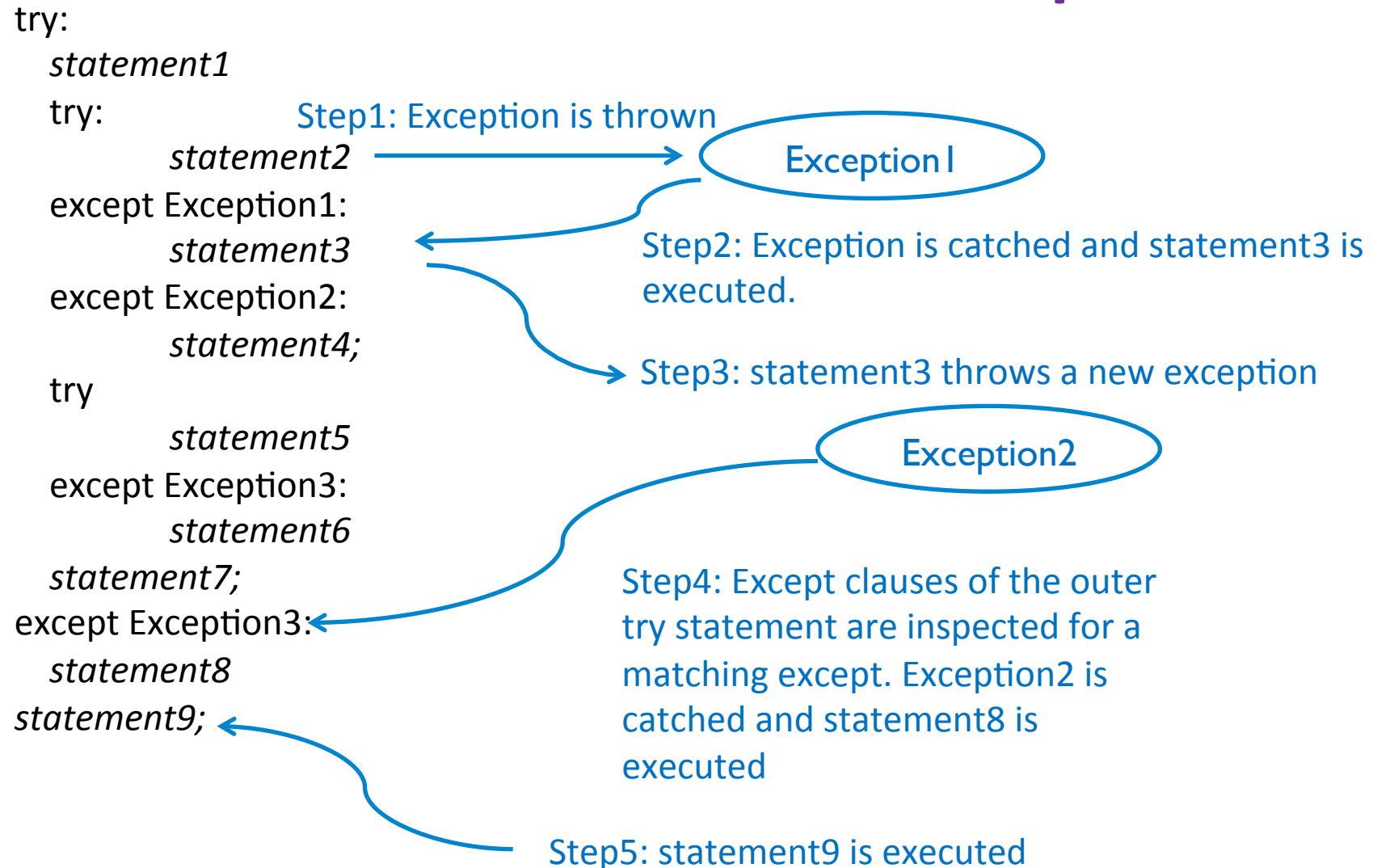
Scenario: statement2 throws Exception1



Scenario: statement2 throws Exception3



Scenario: statement2 throws Exception1 and statement3 throws Exception2



raise Statement

- You can raise exceptions by using the raise statement.

```
def myLevel( level ):  
    if level < 1: raise "Invalid level!", level  
    # The code below to this would not be executed  
    # if we raise the exception  
  
try:  
    call to the function myLevel (-1)  
except "Invalid level!":  
    Exception handling goes here...
```

Custom Exceptions

- Users can define their own exception by creating a new class in Python.
- This exception class has to be derived, either directly or indirectly, from Exception class.
- Most of the built-in exceptions are also derived from this class.

Custom Exceptions

```
class ValueTooSmallError(Exception):  
    """Raised when the input value is too small"""  
    pass  
  
class ValueTooLargeError(Exception):  
    """Raised when the input value is too large"""  
    pass  
  
number = 10          # you need to guess this number  
  
while True:  
    try:  
        i_num = int(input("Enter a number: "))  
        if i_num < number:  
            raise ValueTooSmallError  
        elif i_num > number:  
            raise ValueTooLargeError  
        break  
    except ValueTooSmallError:  
        print("This value is too small, try again!")  
    except ValueTooLargeError:  
        print("This value is too large, try again!")  
  
print("Congratulations! You guessed it correctly.")
```