

COMPSCI 105 S1 2017 Principles of Computer Science

Exceptions



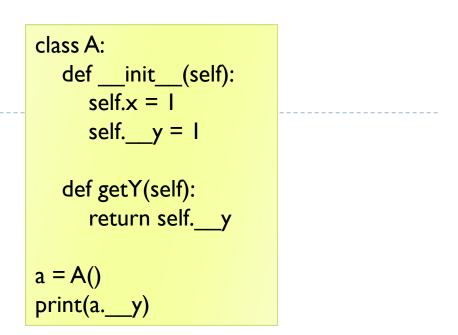
The ______statement causes the __str___ method to be invoked.

- A. print(objectOfClass).
- B. print("object")
- c. objectOfClass.print().
- D. None of the others.

x = Fraction(2, 3)



Consider the following code:



- A. The program has an error because x is private and cannot be access outside of the class.
- B. The program has an error because y is private and cannot be access outside of the class.
- C. The program has an error because you cannot name a variable using ___y.
- D. The program runs fine and prints I.
- E. The program runs fine and prints 0.



- Understand the flow of control that occurs with exceptions
 - try, except, finally
- Use exceptions to handle unexpected runtime errors gracefully
 - 'catching' an exception of the appropriate type
- Generate exceptions when appropriate
 - raise an exception

Resources:

- Errors and Exceptions Python 3.4.2 documentation
 - https://docs.python.org/3/tutorial/errors.html
- Python3 Tutorial: Exception Handling
 - http://www.python-course.eu/python3_exception_handling.php



- Errors occur in software programs. However, if you handle errors properly, you'll greatly improve your program's readability, reliability and maintainability.
 - Python uses exceptions for error handling
- Exception examples:
 - Attempt to divide by ZERO
 - Couldn't find the specific file to read
- The run-time system will attempt to handle the exception (default exception handler), usually by displaying an error message and terminating the program.

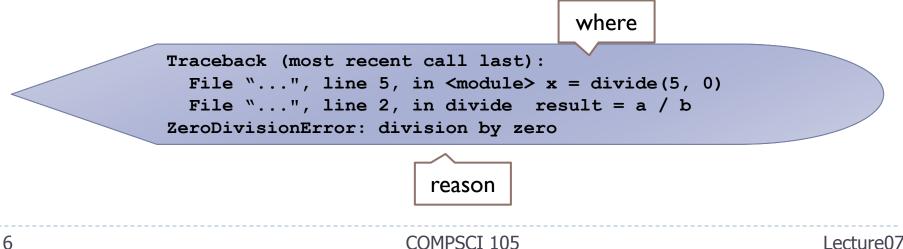


- What if the function is passed a value that causes a divide by zero?
 - Error caused at runtime
 - Error occurs within the function
 - Problem is with the input
- What can we do?

def divide(a, b):
 result = a / b
 return result

```
x = divide(5, 0)
print(x)
```







Check for valid input first

Only accept input where the divisor is <u>non-zero</u>

```
def divide(a, b):
    if b == 0:
        result = 'Error: cannot divide by zero'
    else:
        result = a / b
    return result
```

What if "b" is not a number?

```
def divide(a, b):
    if (type(b) is not int and
        type(b) is not float):
            result = "Error: divisor is not a number"
    elif b == 0:
        result = 'Error: cannot divide by zero'
...
```



Check for valid input first

What if "a" is not a number?



- An exception is an event that occurs during the execution of a program that <u>disrupts</u> the normal flow of instructions during the execution of a program.
- When an error occurs within a method, the method creates an exception object and hands it off to the runtime system.
- The exception object contains
 - information about the error, including its <u>type</u> and the <u>state</u> of the program when the error occurred.
- Creating an exception object and handing it to the runtime system is called <u>throwing an exception</u>.

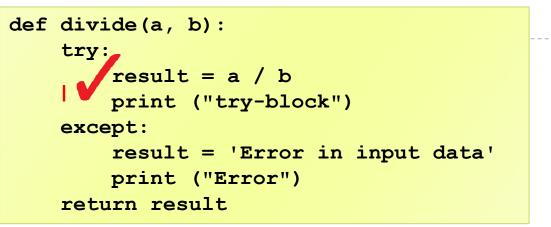


- Code that might create a runtime error is enclosed in a try block
 - Statements are executed sequentially as normal
 - If an error occurs then the remainder of the code is <u>skipped</u>
 - The code <u>starts executing</u> again at the except clause
 - The exception is "caught"

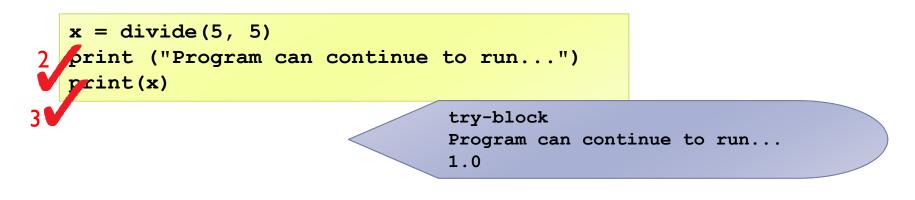
```
try:
    statement block
    statement block
    statement block
except:
    exception handling statements
    exception handling statements
```

- Advantages of catching exceptions:
 - It allows you to fix the error
 - It prevents the program from automatically terminating





- Case I: No error
 - b divide(5,5)





```
def divide(a, b):
    try:
        result = a / b
        print ("try-block"))
    except:
        result = 'Error in input data'
        print ("Error")
    return result
```

- Case 2: Invalid input
 - divide(5,0)
 - divide(5, 'Hello')

x = divide(5, 'hello')
print ("Drogram can contin

Error Program can continue to run... Error in input data

- But what is the error in each situation?
 - I) 5/0 => ZeroDivisionError: division by zero
 - > 2) 5/'hello' =>TypeError: <u>unsupported</u> operand type(s) for /: 'int' and 'str'

3



What is the output of the following?

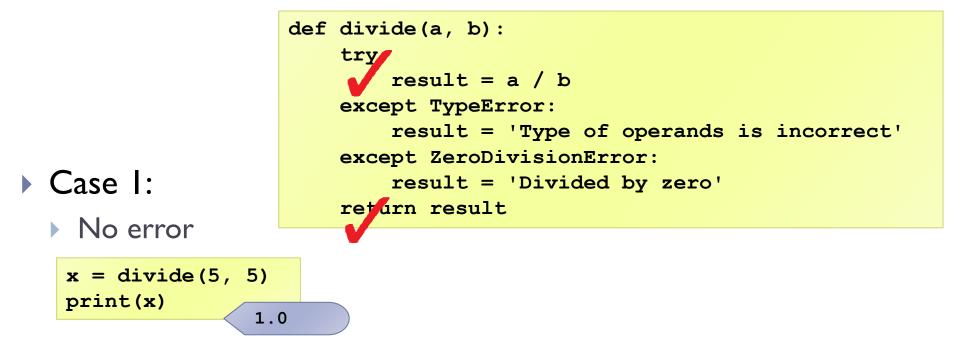
```
def divide(dividend, divisor):
    try:
        quotient = dividend / divsor
    except:
        quotient = 'Error in input data'
    return quotient

x = divide(5, 0)
print(x)
x = divide('hello', 'world')
print(x)
x = divide(5, 5)
print(x)
```

Danger in catching all exceptions

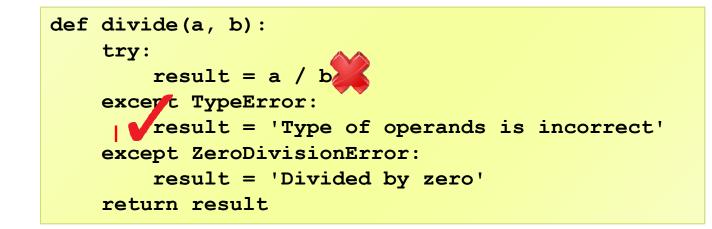
- The general <u>except</u> clause catching <u>all</u> runtime errors
 - Sometimes that can hide problems
- You can put two or more except clauses, each except block is an exception handler and handles the type of exception indicated by its argument in a program.
 - The runtime system invokes the exception handler when the handler is the <u>FIRST ONE</u> matches the <u>type</u> of the exception thrown.
 - It executes the statement inside the matched except block, the other except blocks are bypassed and continues after the try-except block.



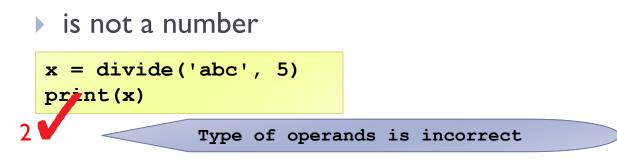




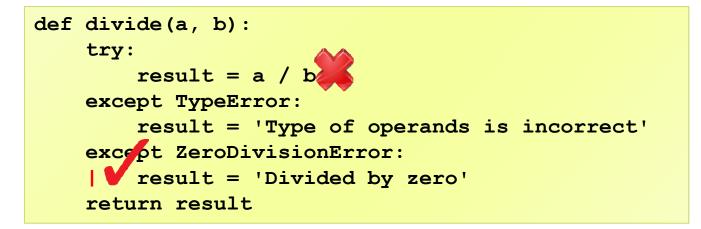




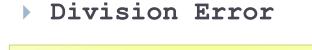
Case 2:

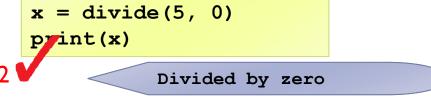






• Case 3:

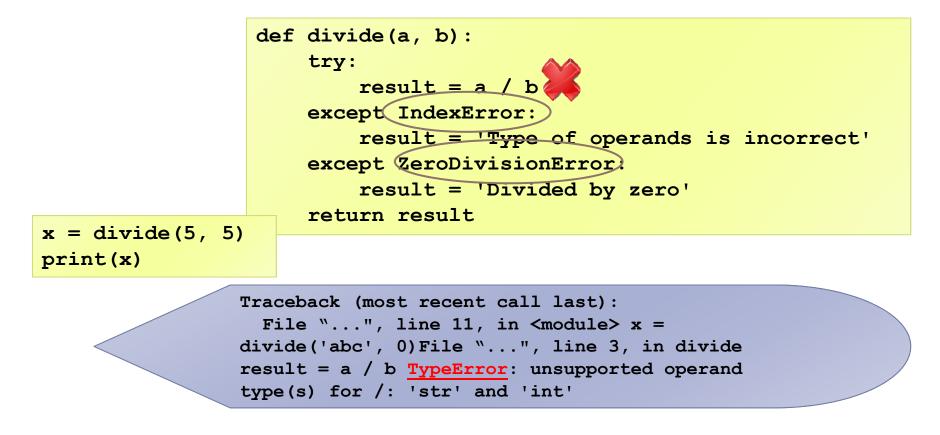




Example03.py

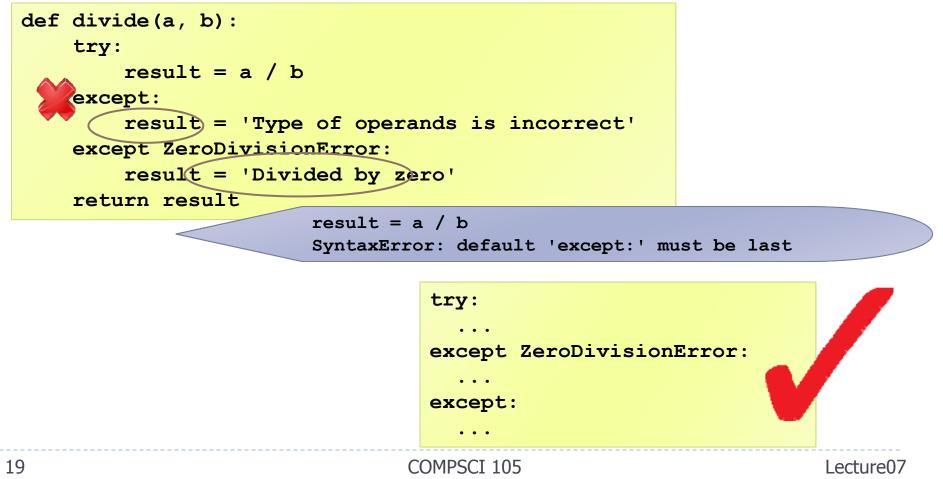


If no matching except block is found, the run-time system will attempt to handle the exception, by terminating the program.





 Specific exception block must come before any of their general exception block





Any kind of built-in error can be caught

- Check the Python documentation for the complete list
- Some popular errors:
 - ArithmeticError: various arithmetic errors
 - ZeroDivisionError
 - IndexError: a sequence subscript is out of range
 - TypeError: inappropriate type
 - ValueError:
 - \square has the right type but an inappropriate value
 - IOError: Raised when an I/O operation
 - EOFError:

□ hits an end-of-file condition (EOF) without reading any data --- ReferenceFrom

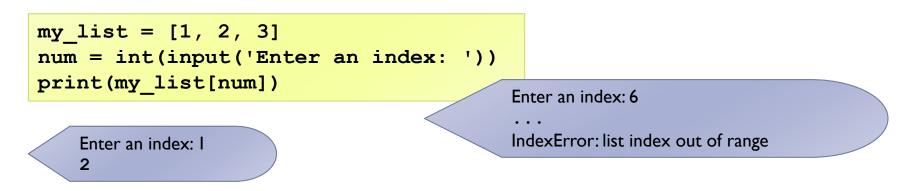
COMPSCI 105

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
   +-- StopIteration
   +-- ArithmeticError
     +-- FloatingPointError
     +-- OverflowError
     +-- ZeroDivisionError
   +-- AssertionError
   +-- AttributeError
   +-- BufferError
   +-- EOFError
   +-- ImportError
   +-- LookupError
     +-- IndexErro
     +-- KeyError
   +-- MemoryError
   +-- NameError
   | +-- UnboundLocalError
   +-- OSError
     +-- BlockinglOError
      +-- ChildProcessError
      +-- ConnectionError
         +-- BrokenPipeError
         +-- ConnectionAbortedError
         +-- ConnectionRefusedError
         +-- ConnectionResetError
      +-- FileExistsError
      +-- FileNotFoundErro
      +-- InterruptedError
      +-- IsADirectoryError
      +-- NotADirectoryError
      +-- PermissionError
      +-- ProcessLookupError
     +-- TimeoutError
   +-- RuntimeError
     +-- NotImplementedError
   +-- SyntaxError
     +-- IndentationError
         +-- TabError
   +-- SystemError
   +-- TypeError
   +-- ValueError
     +-- UnicodeError
                                       Lecture07
         +-- UnicodeDecodeError
         +-- UnicodeEncodeError
         +-- UnicodeTranslateError
```

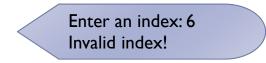
. . .



Consider the following code:

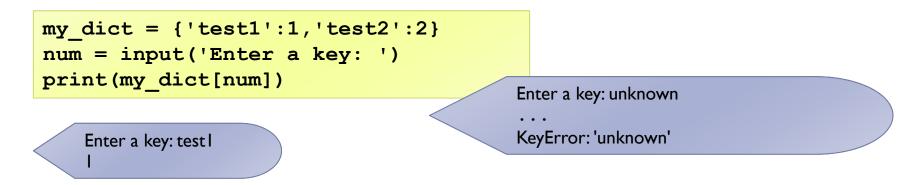


Rewrite it using try-except block to handle the IndexError

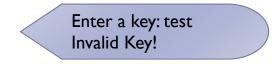




Consider the following code:



Rewrite it using try-except block to handle the KeyError





If you want to give the user more specific feedback about which input was wrong

```
try:
    dividend = int(input("Please enter the dividend: "))
    divisor = int(input("Please enter the divisor: "))
    print("%d / %d = %f" % (dividend, divisor, dividend/divisor))
except ValueError:
    print("The divisor and dividend have to be numbers!")
except ZeroDivisionError:
    print("The dividend may not be zero!")
```

```
try:
    dividend = int(input("Please enter the dividend: "))
except ValueError:
    print("The dividend has to be a number!")
try:
    divisor = int(input("Please enter the divisor: "))
except ValueError:
    print("The divisor has to be a number!")
try:
    print("%d / %d = %f" % (dividend, divisor, dividend/divisor))
except ZeroDivisionError:
    print("The dividend may not be zero!")
```