Exceptions and Exception Handling

Lesson Objectives

At the end of this lesson, you should be able to:

- Explain the concepts of exceptions and exception handling
- Apply exception handling in Python

Topic Outline

Why & What

`try`/`except` Group

Exception Handling Philosophy

`else`/`finally` Group

Why do we need exception handling?

- Most modern languages provide ways to deal with "exceptional" situations
- Dealing with problems
- To try to capture certain situations/failures and deal with them gracefully
- All about being a good programmer!

What counts as an exception?

- Errors
  - indexing past the end of a list
  - trying to open a nonexistent file
  - fetching a nonexistent key from a dictionary, etc.
- Events (not really errors)
  - Search algorithm doesn’t find a value
  - Mail message arrives, queue event occurs

Example - Bad Input

In general, we assume that the input we receive (from a file or from the user) is correct.

This is almost never true. There is always the chance that the input could be wrong.

Our programs should be able to handle this.

"All input is evil until proven otherwise."

- "Writing Secure Code", by Howard and Leblanc

Try/Except Group

General Idea

1. Keep watching a particular section of code

2. If we get an exception, look for a catcher that can handle that kind of exception

3. If found, handle it

4. Otherwise, let Python handle it (which usually halts the program)

General form

```
try:
    Code to run
except aParticularError:
    Stuff to do on error
```

Try suite

```
try:
    Code to run
```

- The **try** suite contains code that we want to monitor for errors during execution
- If an error occurs anywhere in that **try** suite, Python looks for a handler that can deal with the error
- If no specific handler exists, Python handles it

  → The program halts with an error message (we have seen this so many times ☹)

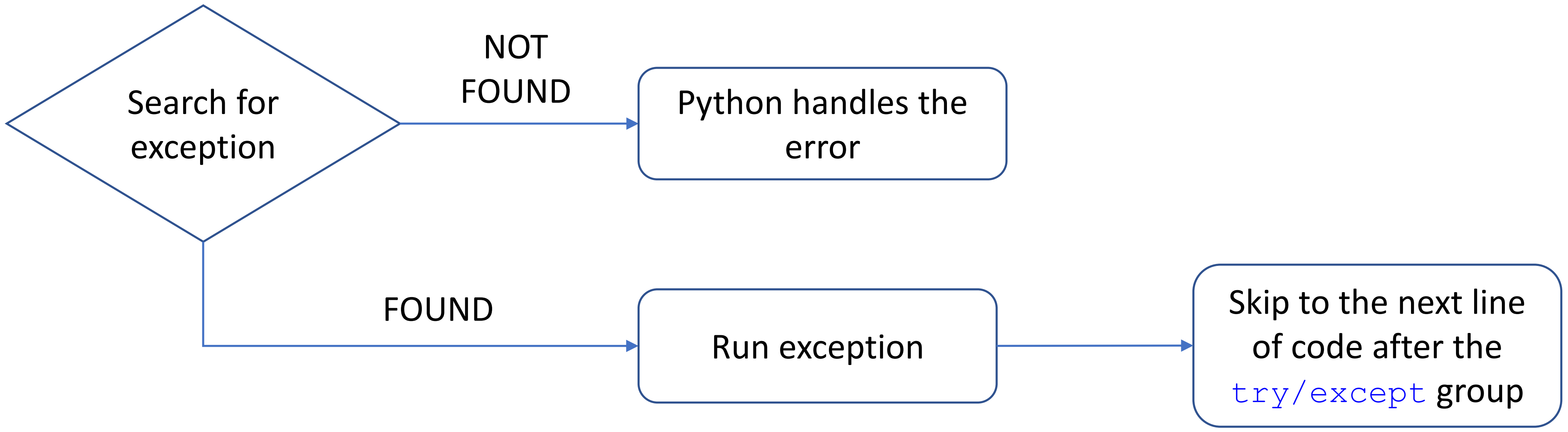
Except suite

```
except aParticularError:
    Stuff to do on error
```

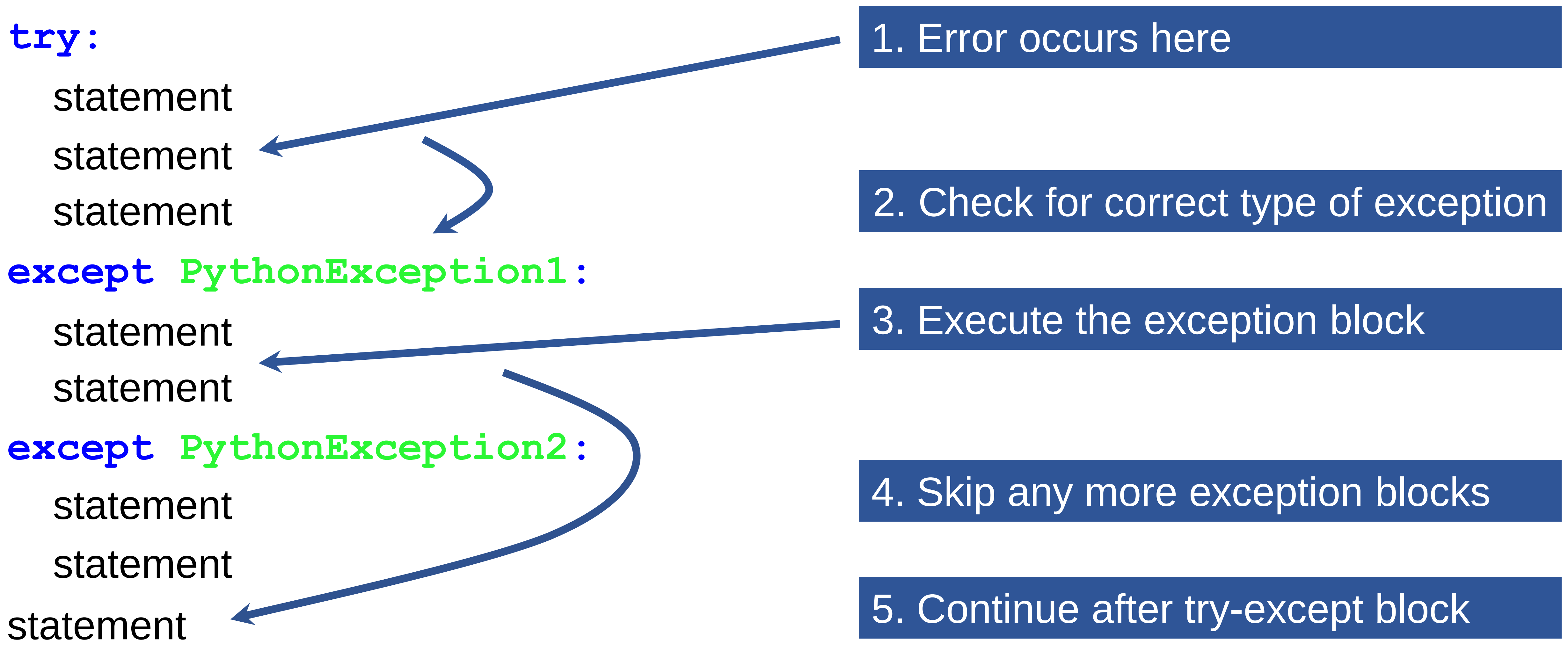
- An `except` suite is associated with a `try` suite
- A `try` suite can have multiple `except` suites
- Each `except` names a type of exception it is monitoring for (can handle)
- If the error occurring in the `try` suite matches the type of exception, then the first `except` suite is activated

Try/Except group

- If no exception is in the try suite, skip to the next line of code after the try/except group
- If an error occurs in a try suite, look for the right exception



Try/Except group



```
try:
    statement
    statement
    statement
except PythonException1:
    statement
    statement
except PythonException2:
    statement
    statement
statement
```

1. Error occurs here
2. Check for correct type of exception
3. Execute the exception block
4. Skip any more exception blocks
5. Continue after try-except block

Try/Except group: An example

Try/Except Group

Try suite

```
try:
      print(“Entering try suite”)
      dividend = float(input(“dividend:”))
      divisor = float(input(“divisor:”))
      result = dividend/divisor
      print(“result = “,result)
```

Except suite 1

```
except ZeroDivisionError:
      print (“Divide by 0 error”)
```

Except suite 2

```
except ValueError:
      print (“Couldn’t convert to a float”)

print (“Continuing with the rest of the program”)
```

Try/Except group: An example – Case 1

```
try:
      print(“Entering try suite”)
      dividend = float(input(“dividend:”))
      divisor = float(input(“divisor:”))
      result = dividend/divisor
      print(“result = “,result)

except ZeroDivisionError:
      print (“Divide by 0 error”)

except ValueError:
      print (“Couldn’t convert to a float”)

print (“Continuing with the rest of the program”)
```

```
dividend: ‘a’

Couldn’t convert to a float

Continuing with the rest of
the program
```

Try/Except group: An example – Case 2

```
try:
      print(“Entering try suite”)
      dividend = float(input(“dividend:”))
      divisor = float(input(“divisor:”))
      result = dividend/divisor
      print(“result = “,result)

except ZeroDivisionError:
      print (“Divide by 0 error”)

except ValueError:
      print (“Couldn’t convert to a float”)

print (“Continuing with the rest of the program”)
```

```
dividend: 4
divisor:  0
4.0/0.0 X

Divide by 0 error

Continuing with the rest of
the program
```

Try/Except group: An example – Case 3

```
try:
      print(“Entering try suite”)
      dividend = float(input(“dividend:”))
      divisor = float(input(“divisor:”))
      result = dividend/divisor
      print(“result = “,result)

except ZeroDivisionError:
      print (“Divide by 0 error”)

except ValueError:
      print (“Couldn’t convert to a float”)

print (“Continuing with the rest of the program”)
```

```
dividend: 4
divisor:  2
4.0/2.0
result = 2.0

Continuing with the rest of
the program
```

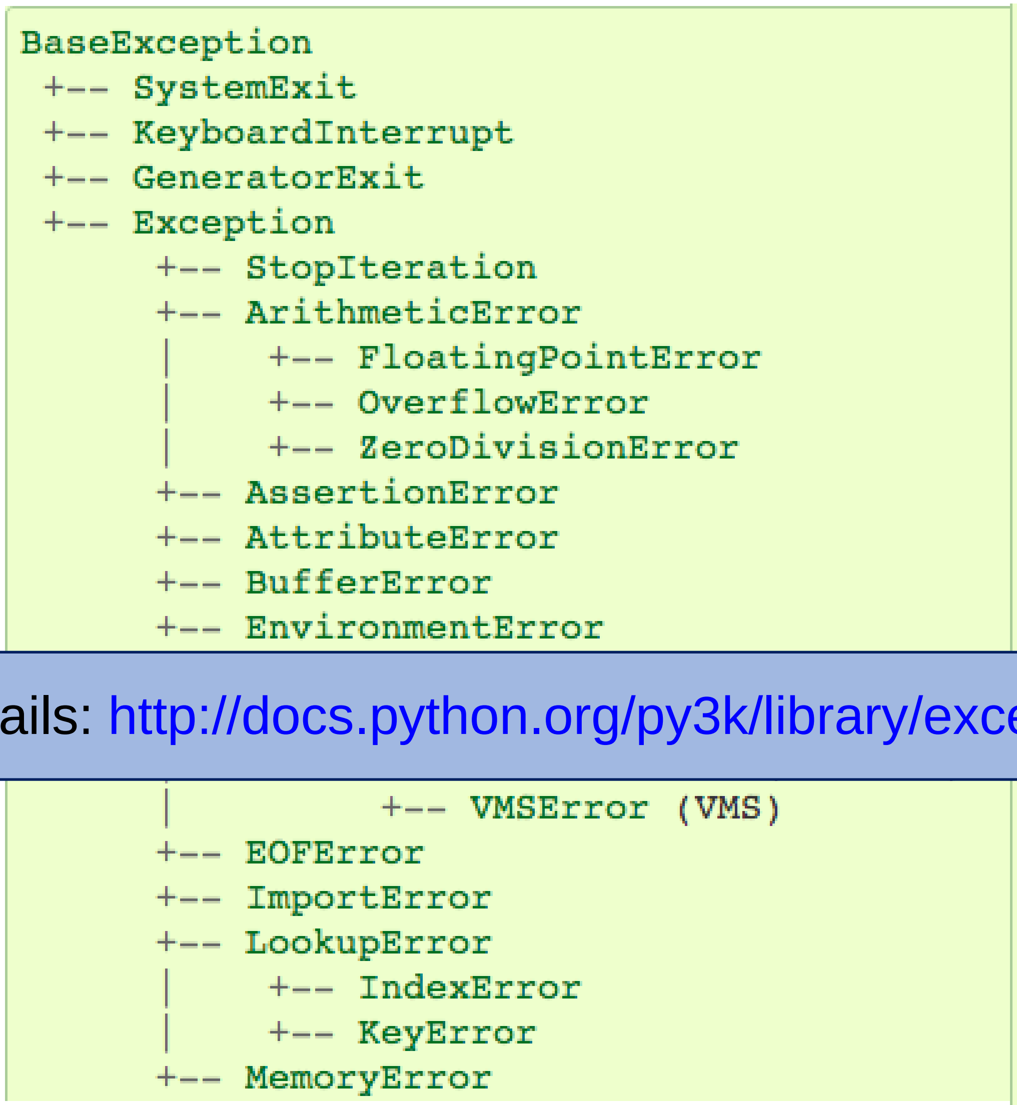
Types of exceptions

- In Python, there is a set of pre-labelled exceptions

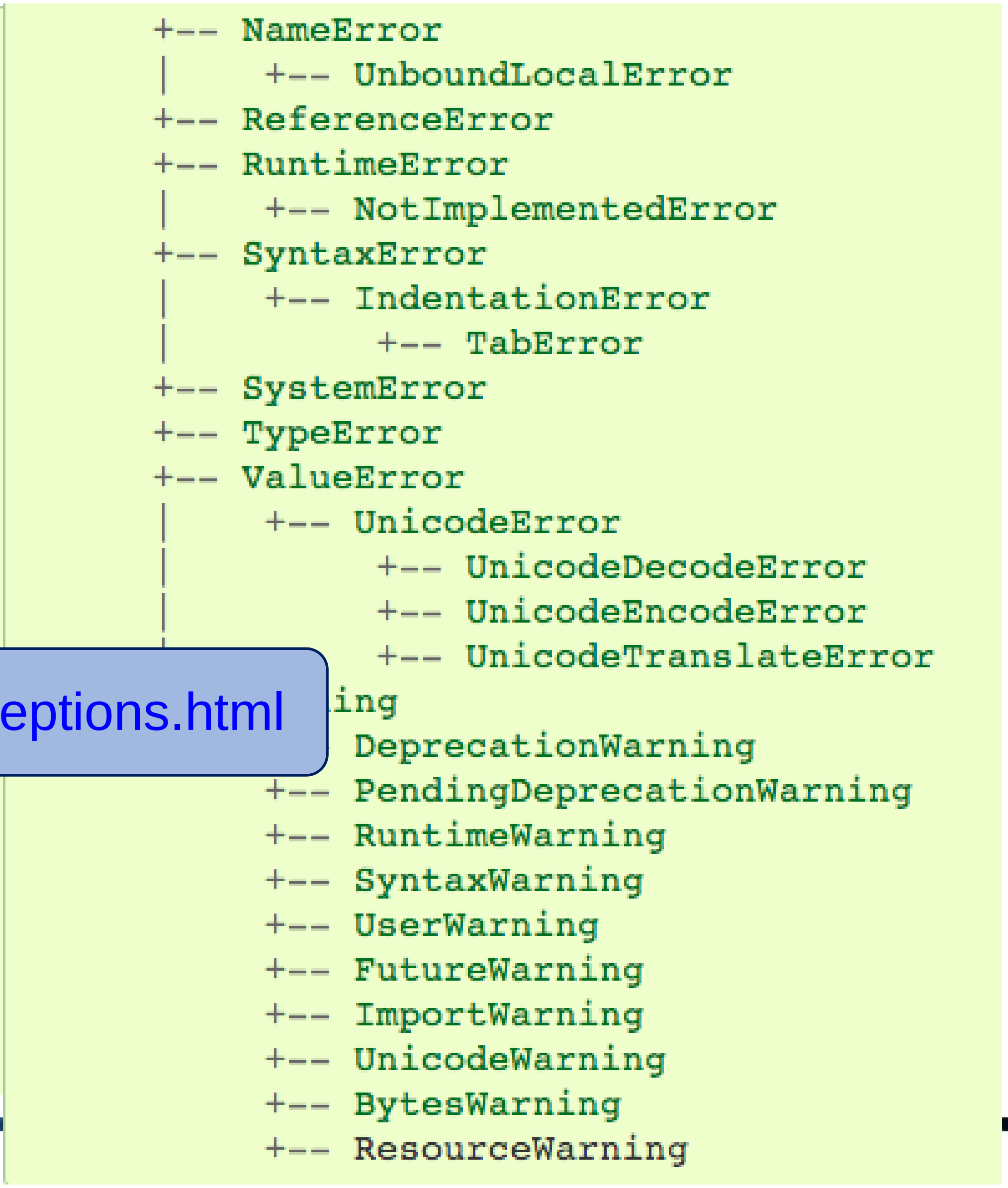
- To find the exception you are interested in, just TRY it in the Python interpreter

```
3/0 =>
ZeroDivisionError: integer division or modulo by zero
```

Types of exceptions (Cont’d)



```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EnvironmentError
      |         +-- VMSError (VMS)
      +-- EOFError
      +-- ImportError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
```



```
      +-- NameError
      |    +-- UnboundLocalError
      +-- ReferenceError
      +-- RuntimeError
      |    +-- NotImplementedError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
      +-- SystemError
      +-- TypeError
      +-- ValueError
      |    +-- UnicodeError
      |         +-- UnicodeDecodeError
      |         +-- UnicodeEncodeError
      |         +-- UnicodeTranslateError
          ing
           DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
           +-- ImportWarning
           +-- UnicodeWarning
           +-- BytesWarning
           +-- ResourceWarning
```

Details: http://docs.python.org/py3k/library/exceptions.html

Exception Handling Philosophy

How you deal with problems

Two ways to deal with exceptions:

LBYL: **L**ook **B**efore **Y**ou **L**eap

EAFP: **E**asier to **A**sk **F**orgiveness than **P**ermission (famous quote by Grace Hopper)

Look Before You Leap

- Be very cautious!
- Check all aspects before execution
  - If string is required: check that
  - If values should be positive: check that
- What happens to length of code?
  - Readability of code - bad

Easier to Ask Forgiveness than Permission

- Run anything you like!
- Be ready to clean up in case of error
- The `try` suite code reflects what you want to do, and the `except` code reflects what you want to do on error
- Cleaner separation!

It’s Your Choice

LBYL

```
if not isinstance(s, str):
      return None
elif not s.isdigit():
      return None
else:
      return int(s)
```

EAFP

```
try:
      return int(s)
except (TypeError, ValueError, OverflowError):
      return None
```

- Python programmers support the EAFP approach:
  - Run the code (in try) and use except suites to deal with errors (don’t check first)

Other Suites

Else suite

- The **`else`** suite is used to execute specific code when no exception occurs

```
try:
        Code to run
except aParticularError:
        Stuff to do on error
else:
        Stuff to do when there is no error
```

Finally suite

- The **`finally`** suite is used to execute code at the end of try/except group (with or without error)

```
try:
    Code to run
except aParticularError:
    Stuff to do on error
finally:
    Stuff to do always at end
```

All together

```
def func(m,n):
        try:
                result = m / n

        except ZeroDivisionError:
                print("Error!")

        else:
                print(result)

        finally:
                print("Goodbye!")
```

```
func(2,0)
```

```
Error!
Goodbye!
```

```
func(2,1)
```

```
2.0
Goodbye!
```

All together (Cont'd)

```
def func(m,n):
      try:
            result = m / n

      except ZeroDivisionError:
            print("Error!")

      finally:
            print("Goodbye!")

      else:
            print(result)
```

Invalid syntax! →

All together (Cont’d)

```
def func(m,n):
      try:
          result = m / n

      except ZeroDivisionError:
          print(“divided by zero”)

      except:
          print(“Error!!”)

      else:
          print(result)

      finally:
          print(“Goodbye!”)
```

No exception name, ok! $\longrightarrow$

```
func(2,0)

divided by zero
Goodbye!
```

```
func(2,’a’)

Error!!
Goodbye!
```

Summary

In this lesson, we have learned:

- The concepts of Exception and Exception Handling
- Exception Handling in Python