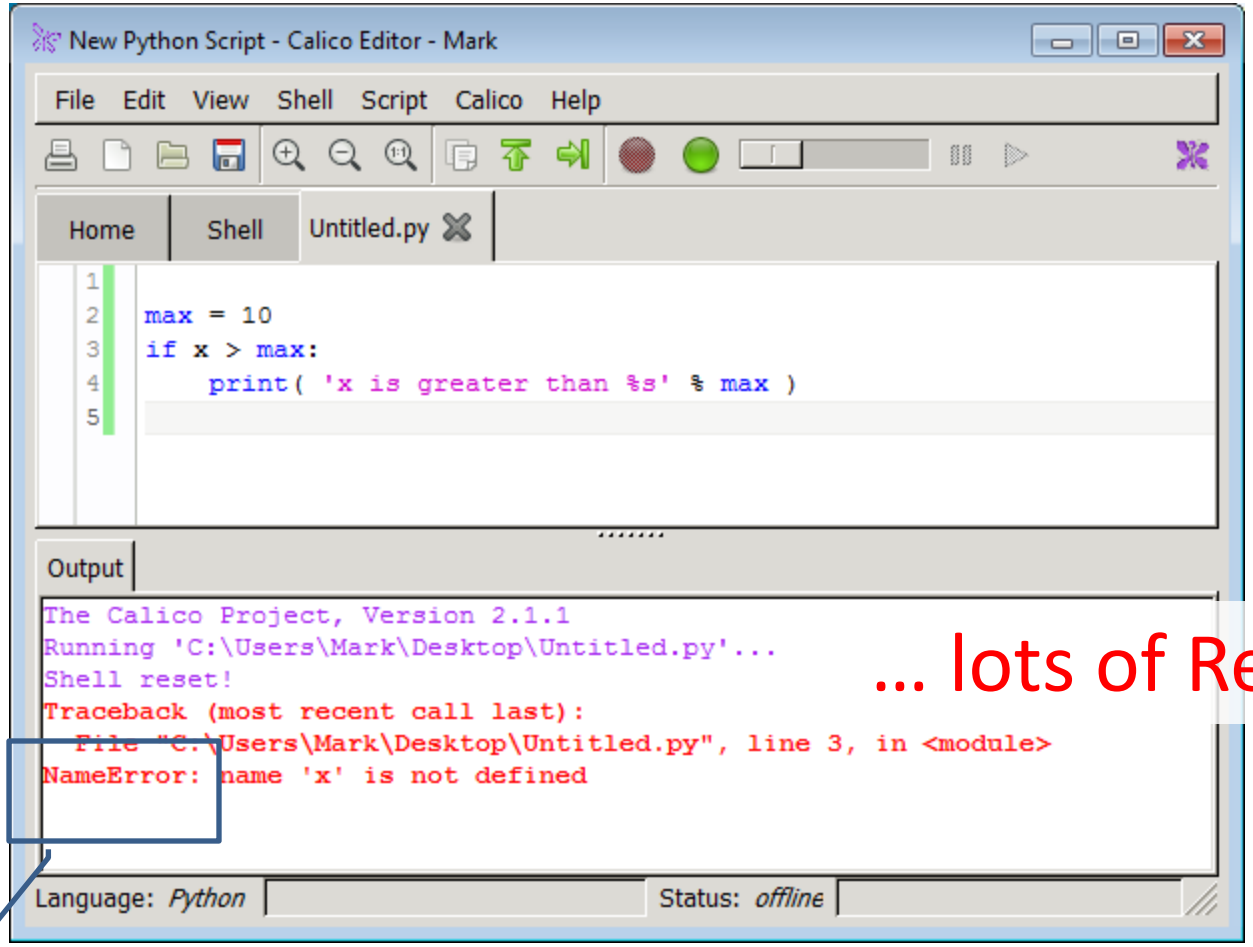
When something goes wrong…



… lots of Red Text

What this? NameError

Exceptions

- It is possible to try to execute a block of code
- but catch any runtime exception, if raised
- and react to the exception programmatically

- NameError is one kind of exception in Python

http://docs.python.org/2/library/exceptions.html

Built-in Exceptions

Raised by Python, including:

ArithmeticError	: mathematical mistake
IndexError	: problem with list index
KeyError	: problem with dictionary key
IOError	: problem with files
EOFError	: tried to read past end of file
FloatingPointError	: numerical mistake
ImportError	: problem with module
NameError	: unassigned variable name
SyntaxError	: statement syntax problem
IndentationError	: code format problem
TypeError	: wrong types for an operator
ValueError	: wrong value for an operation
ZeroDivisionError	: tried to divide by 0
RuntimeError	: other, miscellaneous

… and so many more

Exceptions are Objects – in a hierarchy

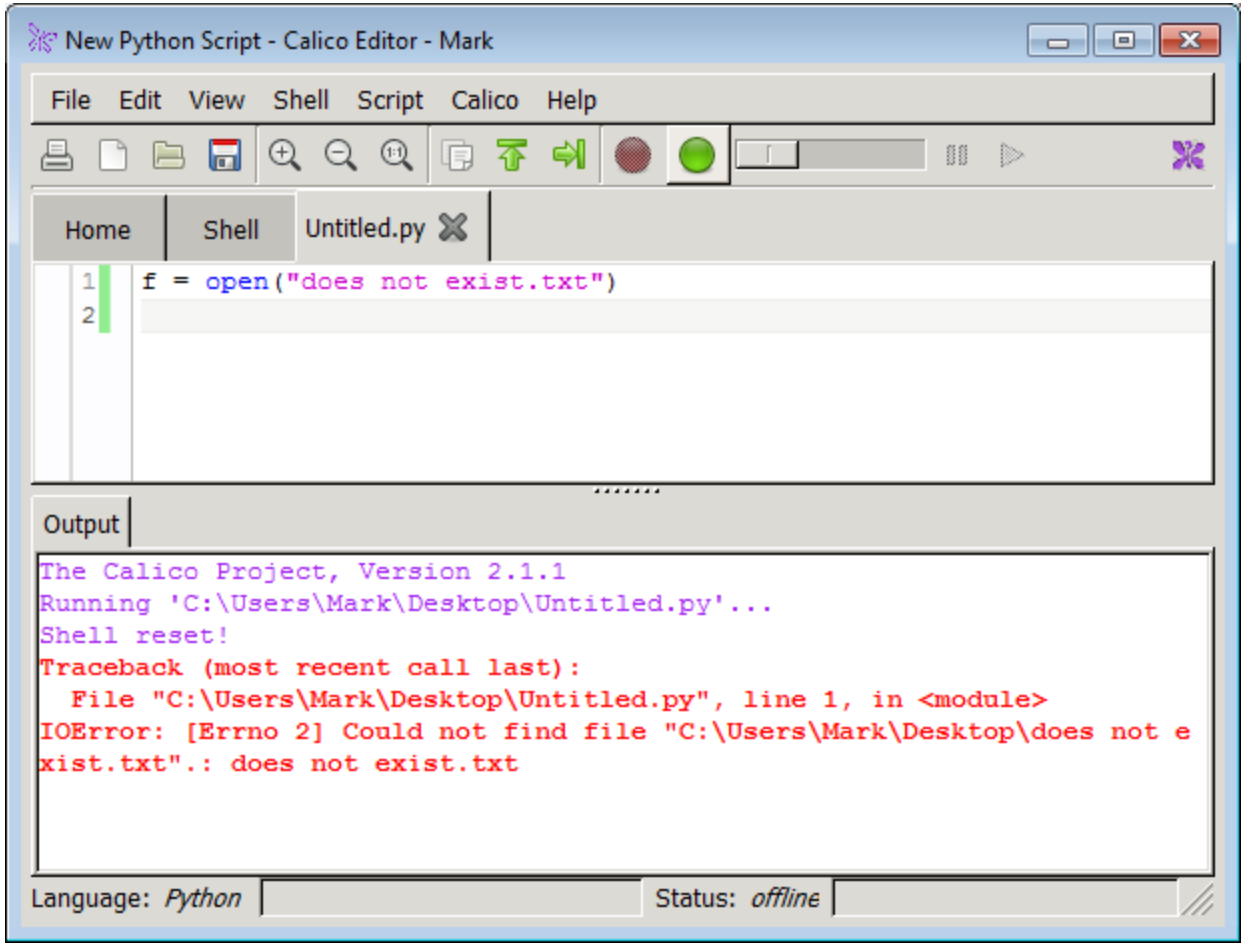
```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StandardError
      |    +-- BufferError
      |    +-- ArithmeticError
      |    |    +-- FloatingPointError
      |    |    +-- OverflowError
      |    |    +-- ZeroDivisionError
      |    +-- AssertionError
      |    +-- AttributeError
      |    +-- EnvironmentError
      |    |    +-- IOError
      |    |    +-- OSError
      |    |         +-- WindowsError (Windows)
      |    |         +-- VMSError (VMS)
      |    +-- EOFError
      |    +-- ImportError
      |    +-- LookupError
      |    |    +-- IndexError
      |    |    +-- KeyError
      |    +-- MemoryError
      |    +-- NameError
      |    |    +-- UnboundLocalError
      |    +-- ReferenceError
      |    +-- RuntimeError
      |    |    +-- NotImplementedError
      |    +-- SyntaxError
      |    |    +-- IndentationError
      |    |         +-- TabError
      |    +-- SystemError
      |    +-- TypeError
      |    +-- ValueError
      |         +-- UnicodeError
      |              +-- UnicodeDecodeError
      |              +-- UnicodeEncodeError
      |              +-- UnicodeTranslateError
      +-- Warning
           +-- DeprecationWarning
           +-- PendingDeprecationWarning
           +-- RuntimeWarning
           +-- SyntaxWarning
           +-- UserWarning
           +-- FutureWarning
             +-- ImportWarning
             +-- UnicodeWarning
             +-- BytesWarning
```

try … Statement

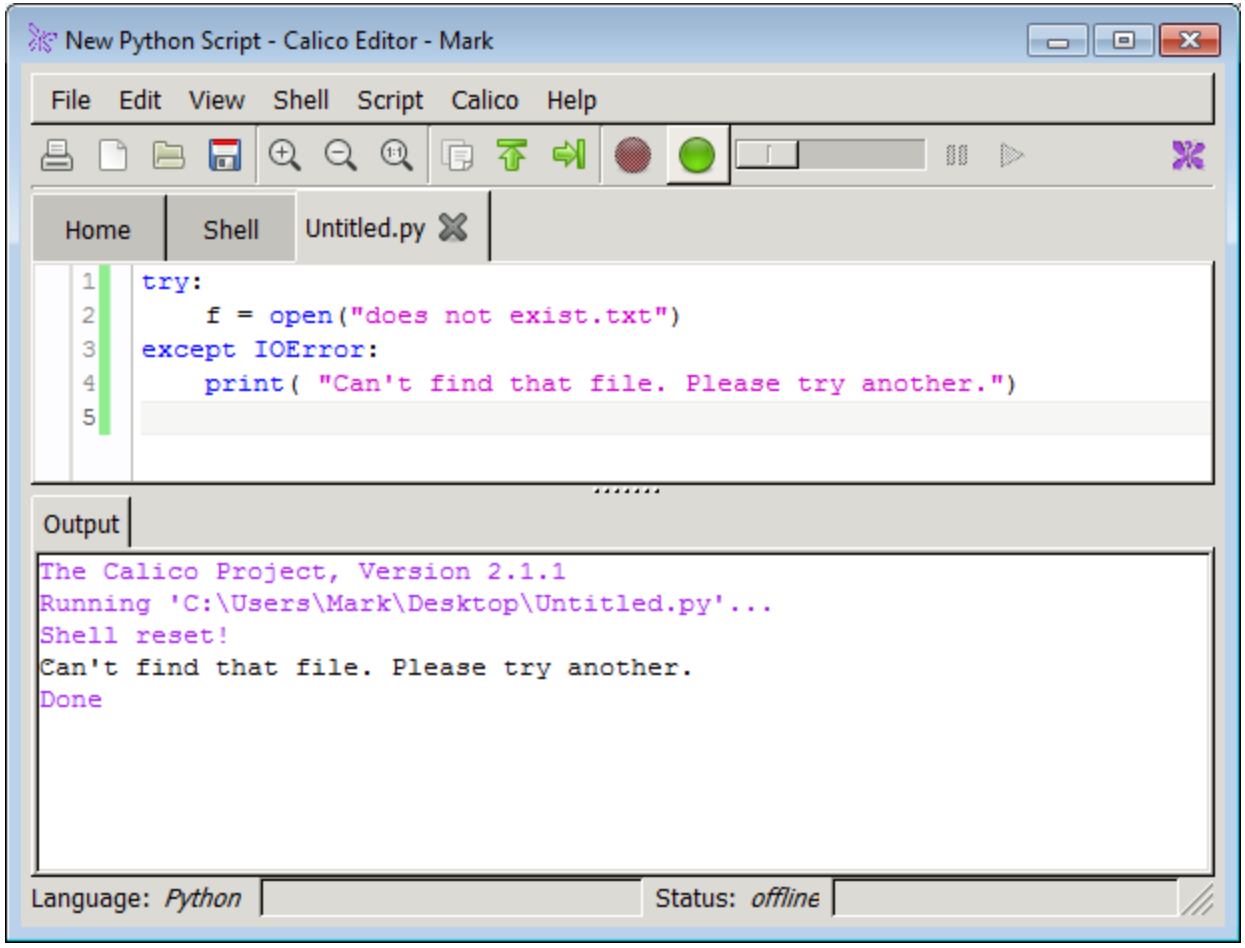
- To catch an exception at runtime, use the try … except statement

```
try:
    # Block of code

except ExceptionClass:
    # Handle the exception with custom code
```

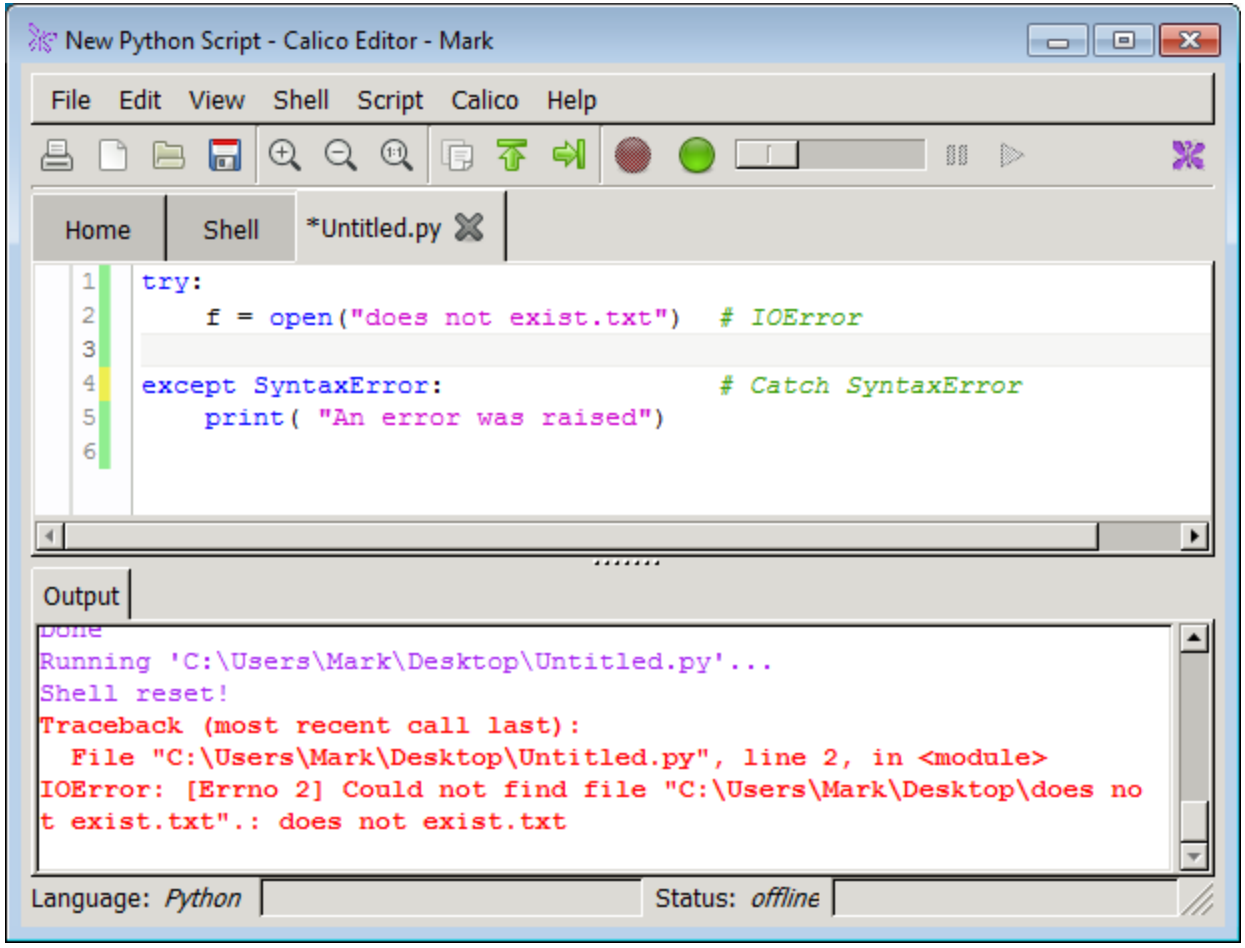
try … except


New Python Script - Calico Editor - Mark
File Edit View Shell Script Calico Help
Home Shell Untitled.py
1 f = open("does not exist.txt")
2
Output
The Calico Project, Version 2.1.1
Running 'C:\Users\Mark\Desktop\Untitled.py'...
Shell reset!
Traceback (most recent call last):
File "C:\Users\Mark\Desktop\Untitled.py", line 1, in <module>
IOError: [Errno 2] Could not find file "C:\Users\Mark\Desktop\does not exist.txt".: does not exist.txt
Language: Python
Status: offline

try … except


New Python Script - Calico Editor - Mark
File Edit View Shell Script Calico Help
Home Shell Untitled.py
1 try:
2 f = open("does not exist.txt")
3 except IOError:
4 print("Can't find that file. Please try another.")
5
Output
The Calico Project, Version 2.1.1
Running 'C:\Users\Mark\Desktop\Untitled.py'...
Shell reset!
Can't find that file. Please try another.
Done
Language: Python
Status: offline

Exception class must match raised exception



Catch multiple errors with a base class

```
try:
    f = open("does not exist.txt")   # IOError

    x = 1/0                          # ZeroDivisionError

    lst = ['a', 'b', 'c']            # IndexError
    y = lst[4]

    dct = {'a':1, 'b':2, 'c':3}      # KeyError
    z = dct['d']

    print( a )                       # NameError

except StandardError:
    print( "An standard error was raised")
```

Why does this work? Think about what the inheritance relationship implies.

Catch all errors with no class at all

```
try:
    f = open("does not exist.txt")   # IOError

    x = 1/0                          # ZeroDivisionError

    lst = ['a', 'b', 'c']            # IndexError
    y = lst[4]

    dct = {'a':1, 'b':2, 'c':3}      # KeyError
    z = dct['d']

    print( a )                       # NameError

except:
    print( "An error was raised")
```

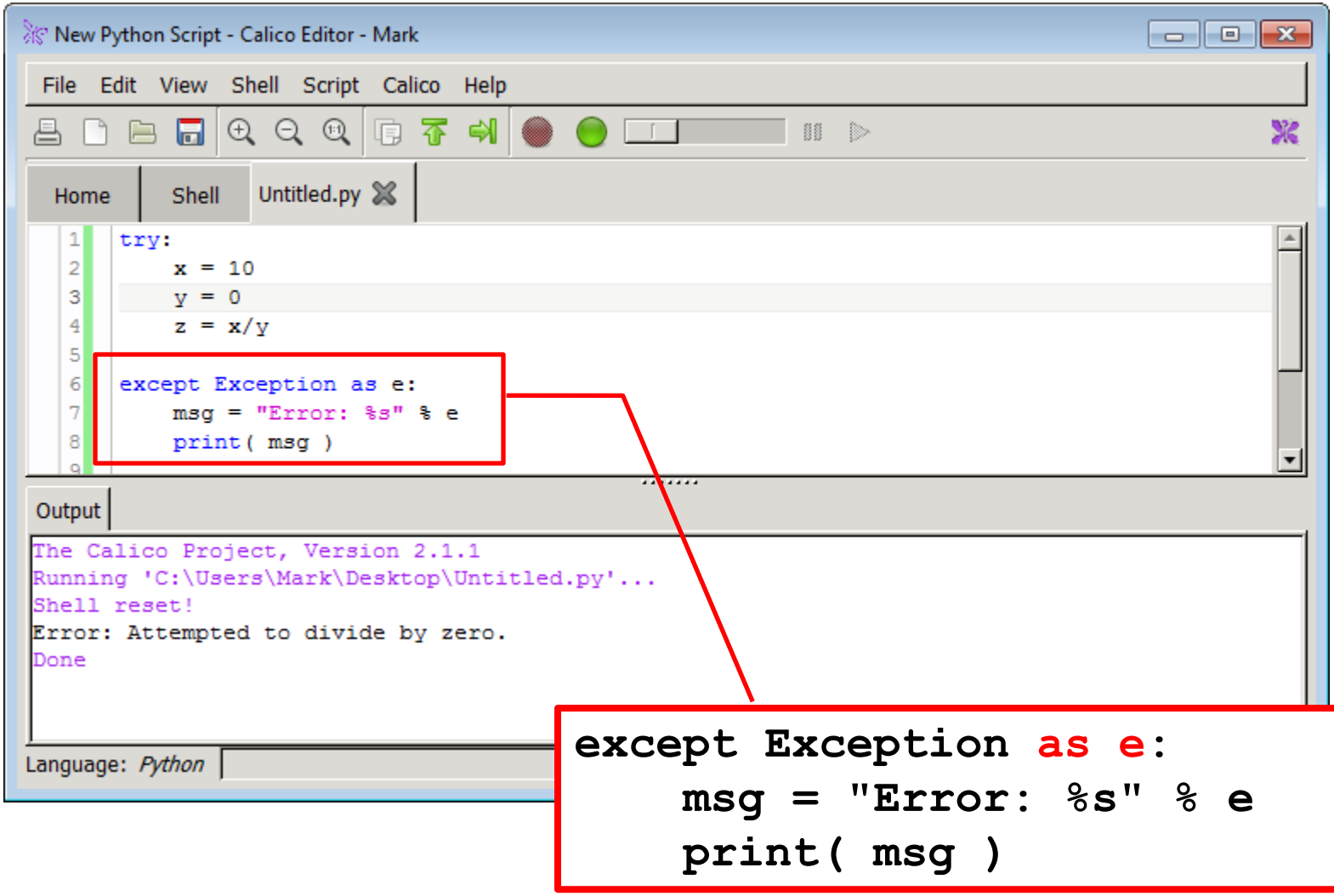
Handle errors with multiple except statements

```
try:
    # Attempt to execute some block of code
    x = 10
    y = 20
    z = x/y

except ZeroDivisionError:
    # Handle case when y = 0
    print( "y cannot be 0" )

except SyntaxError:
    # Handle case when the expression is incorrect
    print( "You have a typo" )

except:
    print( "Something else is wrong" )
```

All try … statement options

```
try:
    # Attempt to execute some block of code
    x = 10
    y = 20
    z = x/y

except ZeroDivisionError:
    # Handle case when y = 0
    print( "y cannot be 0" )

except SyntaxError:
    # Handle case when the expression is incorrect
    print( "You have a typo" )

except:
    print( "Something else is wrong" )

else:       # No exception
    print( "No exception occurred " )

finally:    # Execute regardless of outcome
    print( "Do some cleanup " )
```

Get the Exception object with an “as” keyword



Exam 2 Topics

1. String Manipulation
   - len(), strip(), split(), join()
2. Data Structures
   - lists, dictionary, combinations
3. Transformations
   - pushMatrix(), popMatrix(), translate(), rotate(), scale()
4. Image Processing
   - loadPixels(), updatePixels(), getPixel(), setPixel(), image()
5. Functions
   - defining, calling
6. Debugging
7. Recursive Functions
   - Designing: conditional, base case, recursive function call
8. Inheritance
   - subclassing, methods and instance variable overriding

String Manipulation

```
# What is printed?

colors = "red, green, blue"
colors1 = colors.split(",")

print( len(colors1) )

colors2 = colors.split("\t")

print( len(colors2) )
```

String Manipulation

```
# What is printed?

abbrevs = \
  "new jersey:nj, pennsylvania:pa, new york:ny"

abbrevs1 = abbrevs.split(',')
abbrevs2 = abbrevs1[0].split(':')

print( abbrevs2[0], abbrevs2[1] )
```

String Manipulation

```
# What is printed?
snums = "2,5,9,10"
snums2 = snums.split(',')

i1 = snums2[0]
i2 = snums2[1]
i3 = float( snums2[2] )
i4 = int( snums2[3] )

i1i2 = i1 + i2
i3i4 = i3 + i4

print( i1i2 )
print( i3i4 )
print( type(i1i2) )
print( float( i1i2 ) )
```

String Manipulation

```
# What is printed?

snums = "2, 5, 9, 10"      # Now with spaces
snums2 = snums.split(',')

i1 = snums2[0]
i2 = snums2[1]
i1i2 = i1 + i2
#print( float( i1i2 ) )    # Raises an exception

i1i2 = i1.strip() + i2.strip()
print( float( i1i2 ) )
```

String Manipulation

```
# What is printed?

s = "10, 20, 30"
items = s.split(',')
nums = [ float(i) for i in items ]

sum = 0.0
for n in nums:
    sum += n

print(sum)
```

String Manipulation

```
# What is printed?

names = ["prancer", "dancer", "vixen"]

names1 = ",".join(names)
names2 = ", ".join(names)

print(names1)
print(names2)
```

Data Structures

```
# Assign the first element to variable a,
# second to b, and third to c

lst = ['a', 'b', 'c']


 a = lst[0]
 b = lst[1]
 c = lst[2]
```

Data Structures

```
# Write a for-loop that prints the second
# element of each sublist

lists = [ [1,2,3], [4,5,6], [7,8,9], [10,11,12] ]



for list in lists:
    print( list[1] )
```

Data Structures

```
# Write a statement that prints the third person
# in the 'first' list.

dofls = { 'first' : ['manny', 'moe', 'jack'],
          'second': ['tom', 'dick', 'harry'] }



print( dofls['first'][2] )
```

Data Structures

```
# Print the value in matrix at the keys 'row1', 'col2'
matrix = {'row1': {'col1': 11, 'col2': 12, 'col3': 13},
          'row2': {'col1': 21, 'col2': 22, 'col3': 23},
          'row3': {'col1': 31, 'col2': 32, 'col3': 33} }



print( matrix['row1']['col2'] )
```

Data Structures

```
ds = [{'a':1, 'b':2}, {'b':2, 'a':4}, {'a':10, 'b':12}]

# Print the value of 'a' in the third dict on the ds list
print( ds[2]['a'] )

# Print all values with the 'a' key in all dictionaries
for d in ds:
    print( d['a'] )

# Add up all values with the 'b' key and print the sum
s = 0.0
for d in ds:
    s = s + d['b']
print( s )
```