# COE 302
# Object Oriented Programming with Python

**Akinola Oluwole**

soakinola@abuad.edu.ng

# Content

- We will study exceptions, special error objects that only need to be handled when it makes sense to handle them. In particular, we will cover the following:
  - How to cause an exception to occur
  - How to recover when an exception has occurred
  - How to handle different exception types in different ways
  - Cleaning up when an exception has occurred
  - Creating new types of exception
  - Using the exception syntax for flow control

# Raising exceptions

- There are many different exception classes available, and we can easily define more of our own
- They all have in common is that they inherit from a built-in class called **BaseException**. These exception objects become special when they are handled inside the program's flow of control.

```
>>> print "hello Eni"
   File "<stdin>", line 1
   print "hello world"
   SyntaxError: invalid syntax
```

- This print statement was a valid command in the Python 2 and earlier days, but in Python 3, because print is a function, we have to enclose the arguments in parentheses.
- So, if we type the preceding command into a Python 3 interpreter, we get **SyntaxError**

# Raising Exceptions More Examples

- In addition to SyntaxError, some other common exceptions are shown in the following
- example:

```
>>> x = 5 / 0
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
ZeroDivisionError: int division or modulo by zero
>>> lst = [1,2,3]
>>> print(lst[3])
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
IndexError: list index out of range
```

# Raising Exceptions More Examples

```
>>> lst + 2
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
TypeError: can only concatenate list (not "int") to list
>>> lst.add
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
AttributeError: 'list' object has no attribute 'add'
```

# Raising Exceptions More Examples

```
>>> d = {'a': 'hello'}
>>> d['b']
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
KeyError: 'b'
>>> print(this_is_not_a_var)
Traceback (most recent call last):
File "<stdin>", line 1, in <module>
NameError: name 'this_is_not_a_var' is not defined
```

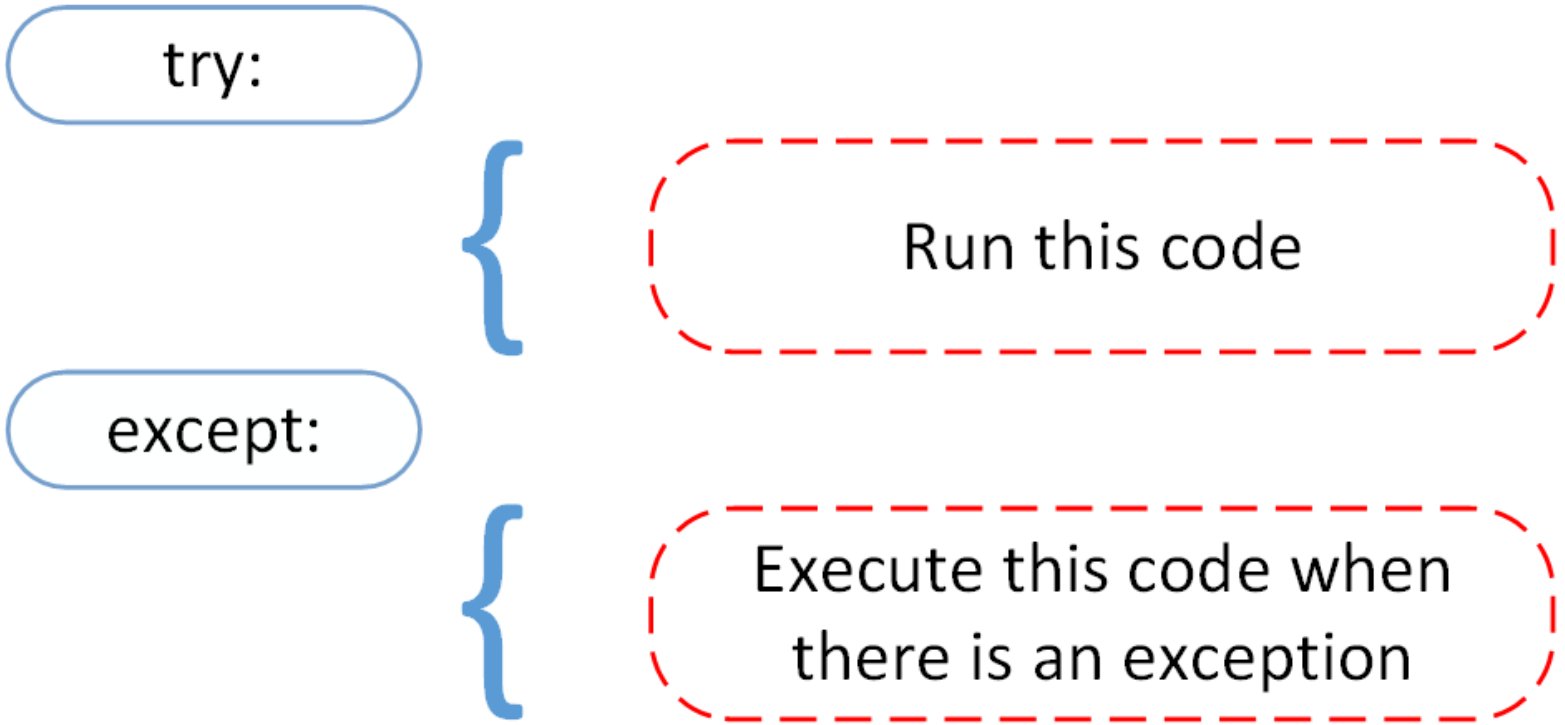
- Sometimes, these exceptions are indicators of something wrong in our program (in which case, we would go to the indicated line number and fix it), but they also occur in legitimate situations.
- The preceding built-in exceptions end with the name Error. In Python, the words error and Exception are used almost interchangeably

# Handling exceptions

- If we encounter an exception situation, how should our code react to or recover from it? We handle exceptions by wrapping any code that might throw one (whether it is exception code itself, or a call to any function or method that may have an exception raised inside it) inside a **try...except** clause.

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")
```

# Handling exceptions



# Raising an exception

- The **try** statement works as follows.
- First, the **try** clause (the statement(s) between the **try** and **except** keywords) is executed.
- If no exception occurs, the **except** clause is skipped and execution of the **try** statement is finished.
- If an exception occurs during execution of the **try** clause, the rest of the clause is skipped. Then if its type matches the exception named after the except keyword, the except clause is executed, and then execution continues after the **try** statement.
- If an exception occurs which does not match the exception named in the except clause, it is passed on to outer **try** statements; if no handler is found, it is an unhandled exception and execution stops with a message.

# Raising an exception

- A class in an except clause is compatible with an exception if it is the same class or a base class thereof (but not the other way around — an except clause listing a derived class is not compatible with a base class). For example, the following code will print B, C, D in that order:

# Raising an exception

```
class B(Exception):
    pass
class C(B):
    pass
class D(C):
    pass
for cls in [B, C, D]:
    try:
        raise cls()
    except D:
        print("D")
    except C:
        print("C")
    except B:
        print("B")
```

# User-defined Exceptions

Programs may name their own exceptions by creating a new exception class. Exceptions should typically be derived from the Exception class, either directly or indirectly.

Exception classes can be defined which do anything any other class can do, but are usually kept simple, often only offering a number of attributes that allow information about the error to be extracted by handlers for the exception. When creating a module that can raise several distinct errors, a common practice is to create a base class for exceptions defined by that module, and subclass that to create specific exception classes for different error conditions:

# User-defined Exceptions

```
class Error(Exception):
    """Base class for exceptions in this module."""
    pass
class InputError(Error):
    """Exception raised for errors in the input.
    Attributes:
        expression -- input expression in which the error occurred
        message -- explanation of the error
    """
    def __init__(self, expression, message):
        self.expression = expression
        self.message = message
```

# User-defined Exceptions

```
class TransitionError(Error):
    """Raised when an operation attempts a state transition that's not
    allowed.
    Attributes:
        previous -- state at beginning of transition
        next -- attempted new state
        message -- explanation of why the specific transition is not
allowed
    """
    def __init__(self, previous, next, message):
        self.previous = previous
        self.next = next
        self.message = message
```

# Reference


- http://www.pythontutor.com/visualize.html#mode=edit
- https://docs.python.org/3/tutorial/errors.html
- Phillips, D. (2010). Python 3 object oriented programming. Packt Publishing Ltd.
- https://realpython.com/python-exceptions/