# COSC 2306
# Data Programming

## OOP and Class

# Exception Handling

- Besides native Python errors, we can raise customized exceptions using the raise statement
  - we specify the name of the error/exception and the exception object
  - the error/exception that we can raise should be a class which is directly or indirectly inherited class of the exception class

# Exception Handling

```
#Define new exception class
class ShortInputException(Exception):
    #A user-defined exception class.
    def __init__(self, length, atleast):
        self.length = length
        self.atleast = atleast
try:
    s = input('Enter something --> ')
    if len(s) < 3:
        raise ShortInputException(len(s), 3)
```

# Exception Handling

except EOFError: #Ctrl-D/Ctrl-Z: no input data is read

    print('\nNo input received (EOF detected).')

except ShortInputException as x:

# **as** assigns the caught exception instance to the variable x, which allows access to any attributes/methods of the exception object x, e.g., attributes *length* and *atleast* here

    print (f"\nThe input was of length {x.length}, it should be at least {x.atleast}")

else:

    print ('No exception was raised.')

# Exception Handling

- What if we wanted some statements to execute after the try block whether or not an exception was raised?
- This is done using the finally block

```
try:
    f = file('poem.txt')
    while True: # Our usual file-reading block
        l = f.readline()
        if len(l) == 0:
            break
        print(l),
finally:
    print ('Cleaning up...')
    f.close()
```

# Exception Handling

- Syntax Error: *a mistake in the structure of a statement or expression*

```
for i in range(10)   # missing ":"
SyntaxError: invalid syntax
```

- Logic Error: *the program executes but gives the wrong result*

```
a_number = int(input("Please enter an integer "))
print(math.sqrt(a_number))

>> Please enter an integer -23
Traceback (most recent call last):
File "<pyshell#102>", line 1, in <module>
print(math.sqrt(a_number))
ValueError: math domain error
```

```
try:
    print(math.sqrt(a_number))
except:
    print("Bad Value for square root")
    print("Using absolute value instead")
    print(math.sqrt(abs(a_number)))

Bad Value for square root
Using absolute value instead
4.795831523312719
```

# Exception Handling

## In-class programming:

Write Python code that takes a user input number, divides 10 by it, and uses **else** to print success and **finally** to always print completion. Note:

- If input is invalid (ValueError), show this message "Invalid input! Enter a valid number."
- If divide zero (ZeroDivisionError), show this message "Cannot divide by zero!"
- If successful, show this message “Division successful! Result =” and the result
- Always have this message “Program execution completed.” at the end

# Exception Handling

```
try:
    value = int(input("Enter a number: "))
    result = 10 / value
except ValueError:
    print("Invalid input! Enter a valid number.")
except ZeroDivisionError:
    print("Cannot divide by zero!")
else:
    print("Division successful! Result =", result)
finally:
    print("Program execution completed.")
```