MTA 98-381 LESSON 8 EXCEPTION HANDLING

3 TYPES OF PROGRAM ERRORS

- 1. Syntax Errors
  - Syntax errors or parsing errors occur when your code does not follow Python syntax (grammar).
  - Examples: missing (:, ', "), etc.
- 2. Exceptions (Runtime Errors)
  - Your program passes Python syntax checking, but crashes during runtime due to a particular runtime error.
  - Examples: divide by zero, file not found, etc.

3 TYPES OF PROGRAM ERRORS (CONT.)

- 3. Logic Errors
  - Your program could run without crashing. However the result is incorrect. Logic errors are usually the most difficult to correct.
  - Examples: wrong formula for calculation, etc.
- Bug – error in code.
- Debug – find the bug and correct it.

HANDLING EXCEPTIONS

- We use `try except finally` blocks to handle exception.
- `try` block contains the code that may trigger exception. If exception occurs then Python jumps to the `except` block.
- `except` block contains the code that handles the exception if it occurs. `except` block won't be executed if there is no exception.
- `finally` block (optional) contains the code that should be executed regardless of exception or not. The code here is usually related to the `try` block.

EXERCISE 1: VALIDATING USER INPUT

- Exception handling is widely used to validate user input, e.g. ensure correct type of user input such as number only.
- 1. Run the following code.

```
number = float(input('Enter a number to square: '))
print (number**2)
```

- 2. Enter 2.5. The program will run without error.
- 3. Enter a. What exception do you get?

EXERCISE 1: VALIDATING USER INPUT (CONT.)

- 4. Modify the code to as follows:

```
try:
  number = float(input('Enter a number to square: '))
  print (number**2)
except:
  print ('Invalid number')
```

- 5. Run the code. Enter 2.5.
  - What is the output?
  - Was the `except` block executed?
- 6. Run the code. Enter a.
  - What is the output?
  - Was the 2nd line in the `try` block executed?

EXERCISE 2: VALIDATING USER INPUT

- Modify the program to continue asking for user input until a valid input is entered.
- Sample output:

```
Enter a number to square: a
Invalid number
Enter a number to square: b
Invalid number
Enter a number to square: c
Invalid number
Enter a number to square: 5
25.0
```

EXERCISE 3: FINALLY BLOCK

- 4. Modify the code to as follows:

```
try:
  number = float(input('Enter a number to square: '))
  print (number**2)
except:
  print ('Invalid number')
finally:
  print ('Bye')
```

- 5. Run the code. Enter 2.5.
  - Was the `finally` block executed?
- 6. Run the code. Enter a.
  - Was the `finally` block executed?

MULTIPLE EXCEPT BLOCKS

- One `try` block may have many `except` blocks as long as each `except` block is designed to handle a unique exception.
- The last `except` block can be generic.
- Example:

```
try:
    ...
except Exception1:
    ...
except Exception2:
    ...
except:           # for exception not specified above.
    ...
```

EXERCISE 4: MULTIPLE EXCEPT BLOCKS

- 1. Run the following code:

```
try:
    choice = int(input ('Enter 1 or 2: '))
    if choice == 1:
        print (x)
    elif choice == 2:
        print (2 + 'a')
except TypeError:
    print ("Data type error")
except NameError:
    print ("Variable name error")
except:
    print ("Unknown error")
```

EXERCISE 4: MULTIPLE EXCEPT BLOCKS (CONT.)

- 2. Run the code. Enter 1.
  - What is the output?
  - Which `except` block exception was executed?
- 3. Run the code. Enter 2.
  - What is the output?
  - Which `except` block exception was executed?
- 4. Run the code. Enter a.
  - What is the output?
  - Which `except` block exception was executed?
- 5. Run the code. Enter 3.
  - What is the output?
  - Which `except` block exception was executed?

IOERROR EXCEPTION

- IOError is typically thrown when there is problem with opening, reading from or writing to a file.

```
try:
    f = open ('file.txt')
except IOError as e:    # e contains the default
    print (e)           # error message.
```