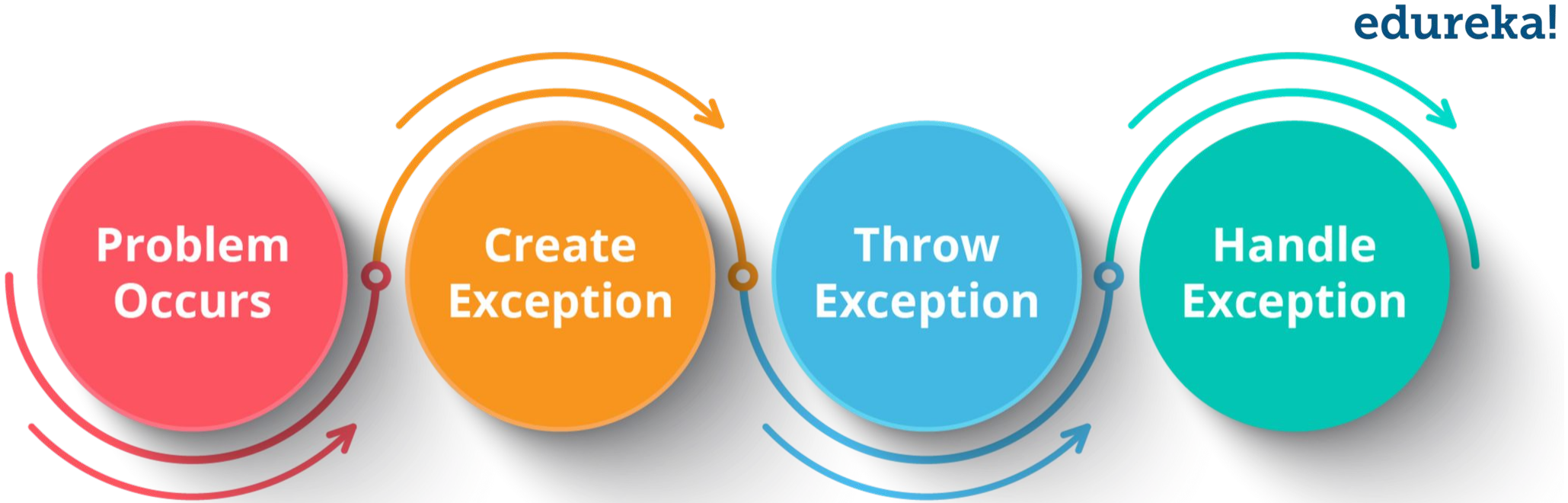
COMP 125 Programming with Python
Exception Handling



Mehmet Sayar

Koç University

Last Time: String Formatting

```
name = 'John'
age = 22
print('My name is %s and I am %d years old' % (name, age))
print('My name is {0} and I am {1} years old'.format(name, age))
print(f'My name is {name} and I am {age} years old’)
```

Output:

```
My name is John and I am 22 years old
My name is John and I am 22 years old
My name is John and I am 22 years old
```

Last Time: String Formatting

```
floatValue = 123.45789
print("Truncate to two digits after decimal in float %.2f" % floatValue)
print("Truncate to two digits after decimal in float {:.2f} ".format(floatValue))
print(f"Truncate to two digits after decimal in float {floatValue:.2f}")
```

Output:

```
Truncate to two digits after decimal in float 123.46
Truncate to two digits after decimal in float 123.46
Truncate to two digits after decimal in float 123.46
```

Last Time: String Formatting

```
floatValue = 123.45789
print("Truncate to two digits after decimal in float %.2e" % floatValue)
print("Truncate to two digits after decimal in float {:.2e} ".format(floatValue))
print(f"Truncate to two digits after decimal in float {floatValue:.2e}")
```

Output:

```
Truncate to two digits after decimal in float 123.46
Truncate to two digits after decimal in float 123.46
Truncate to two digits after decimal in float 123.46
```

Last Time: Parsing

- **Parsing**: The act of reading “raw data” (text or just bytes and converting it into a more useful format stored in memory.

- It involves
  - File reading (or getting the data some other way)
  - String Manipulation (when dealing with text)
  - Control Flow
  - Containers/Collections

Exceptions

- A lot of things can go wrong while coding. For example
  - Trying to open a non-existent file
  - Trying to convert a string with non-digit characters to an integer
  - Trying to access something in a container that does not exist
  - Trying to call a non-existent method of a class
  - Trying to import a non-existent module
  - Trying to modify an immutable object
  - Trying to call a non-existent object
- These are examples of “exceptions”
- Should our programs stop or handle these?

Exceptions

- Exception: indication of an “special event”, usually an error, during program execution
- Exceptions cause the program to abruptly halt, unless handled
- Traceback:
  - Occurs when an exception is encountered
  - Error messages that give information regarding the line numbers that caused the exception
  - Indicates the type of exception and brief description of the error that caused exception to be raised

Exceptions

- Many exceptions can be prevented by careful coding
  - Example: input validation
  - Usually involve a simple decision construct

- Some exceptions cannot be avoided by careful coding. Examples
  - Trying to convert non-numeric string to an integer: Check if the string is made up of digits first
  - Trying to open for reading a file that doesn’t exist: Check if the file exists first

- However, they cannot be always avoided, hence the need to handle them

Exception Handling

- We must first “anticipate” an exception, including its type, write code to “catch” it, and handle it
- We use the `try-except` statements towards this end
- General format:

```
try:
    statements_that_may_cause an exception
except ExceptionType:
    statements_to_handle_the_exception
```

- The code in the except part is sometimes called the exception handler

Exception Handling

```
try:
    statements_that_may_cause an exception
except ExceptionType:
    statements_to_handle_the_exception
```

- If the statement within the try block raises exception:
  - Exception specified in except clause:
    - Handler, immediately following the except clause, executes
    - Continue program after try/except statement
  - Other exceptions:
    - Program halts with traceback error message
- If no exception is raised, handlers (statements in the except block)are skipped

Exception Handling

- Often code in the try block can throw more than one type of exception
- Need to write an except block (aka a handler) for each type of exception that needs to be handled
  - Can specify multiple except blocks
  - Some exceptions may be handled together, just write them as a tuple

- An except clause that does not list a specific exception will handle any exception that is raised in the try block
  - Should always be last in a series of except clauses

An Exception's Error Message

- Exceptions usually come with error messages to help debug the problem
- When an exception is thrown, an exception object (variable) is created, which contains the message
- Can assign the exception object to a variable in an except clause

```
except ValueError as err:
```

- Can pass exception object variable to print function to display the default error message

```
print(err)
```

Using `else` with `try-except`

- `try/except` statement may include an optional `else` clause, which appears after all the `except` clauses
  - Aligned with try and except clauses
- The code within the else block is run when there are no exceptions raised within the except block
- If exception was raised, the else suite is skipped

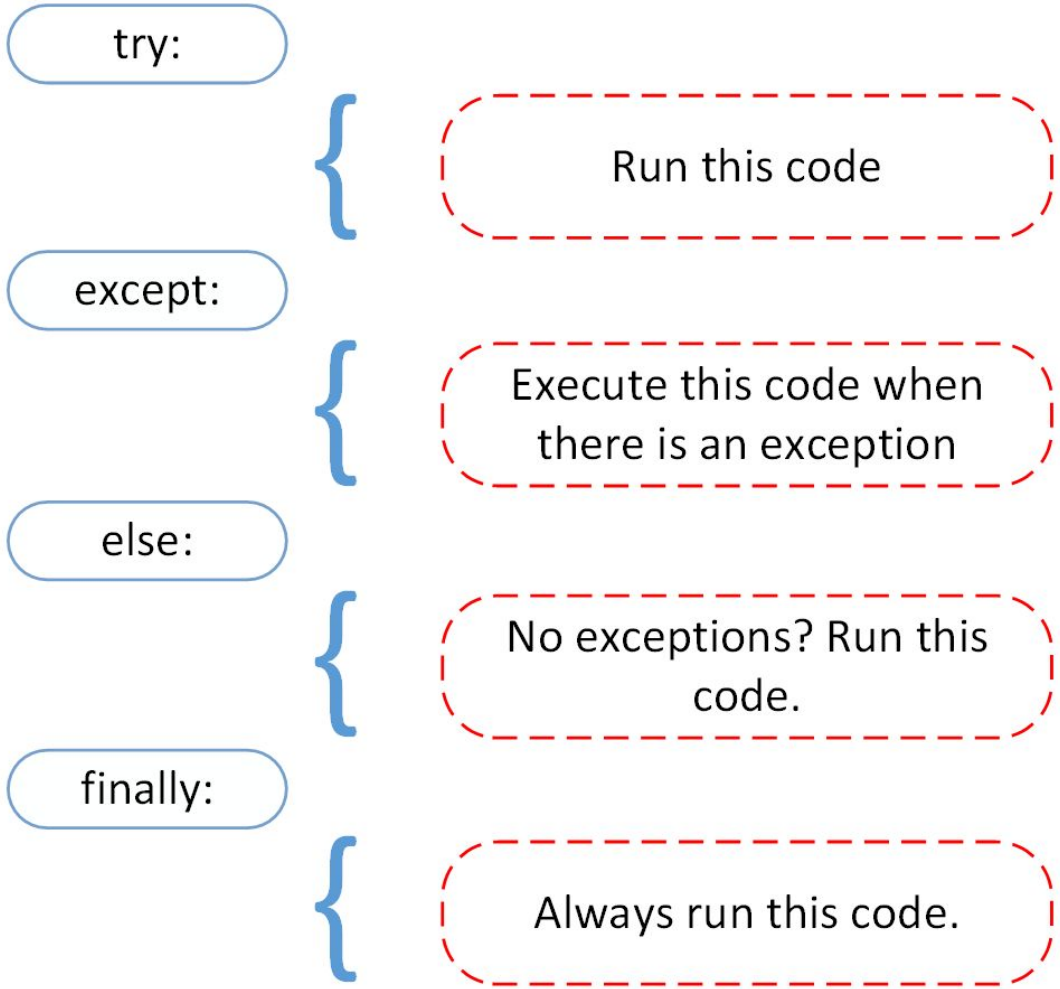
The `finally` Clause

- `try/except` statement may include an optional `finally` clause, which appears after all the `except` clauses
  - Aligned with try and except clauses
- General format:

```
finally:
    statements
```

- The code within the finally block is executed whether an exception occurs or not
- Purpose is to perform cleanup

Summary



Source: Real Python

- Can have multiple except clauses

```
...
except SomeError:
...
except SomeOtherError
...
```

- Can handle multiple exceptions in a single except block

```
...
except (SomeError, SomeOtherError):
...
```

- Can get the exception object with as

```
...
except (SomeError, SomeOtherError) as err:
...
```

- A lonely catches “everything”

```
except:
...
```

Division Example

- Write a program that will take two numbers as input

- It will print out the operation with all the numbers having 3 significant digits after the decimal point, if successful

- Your program should handle the case when the user enters a string that cannot be converted to a number (`ValueError`)
- Your program should handle the case when the user enters the denominator as 0 (`ZeroDivisionError`)

Division Example

```
try:
    number1 = input( "Enter numerator: " )
    number1 = float( number1 ) #may cause ValueError
    number2 = input( "Enter denominator: " )
    number2 = float( number2 ) #may cause ValueError
    result = number1 / number2 #may cause ZeroDivisionError
except ValueError: #Handling the ValueError
    print ("You must enter two numbers")
except ZeroDivisionError: #Handling the ZeroDivisionError
    print("Attempted to divide by zero")
else: #Run if there are no exceptions
    print(f"{number1:.3f} / {number2:.3f} = {result:.3f}")
finally: #Run regardless
    print("Goodbye!")
```