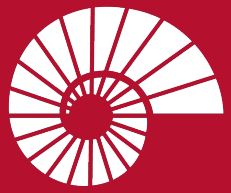


KOÇ
ÜNİVERSİTESİ

11 – Exceptions

COMP 125 Programming with Python

Recording Disclaimer

The synchronous sessions are recorded (audiovisual recordings). The students are not required to keep their cameras on during class.

The audiovisual recordings, presentations, readings and any other works offered as the course materials aim to support remote and online learning. They are only for the personal use of the students. Further use of course materials other than the personal and educational purposes as defined in this disclaimer, such as making copies, reproductions, replications, submission and sharing on different platforms including the digital ones or commercial usages are strictly prohibited and illegal.

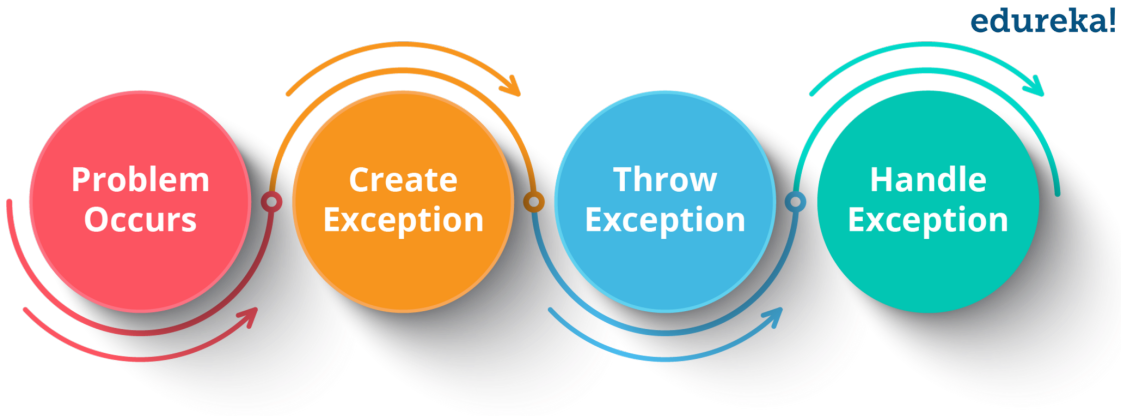
The persons violating the above-mentioned prohibitions can be subject to the administrative, civil, and criminal sanctions under the Law on Higher Education Nr. 2547, the By-Law on Disciplinary Matters of Higher Education Students, the Law on Intellectual Property Nr. 5846, the Criminal Law Nr. 5237, the Law on Obligations Nr. 6098, and any other relevant legislation.

The academic expressions, views, and discussions in the course materials including the audio-visual recordings fall within the scope of the freedom of science and art.

Exceptions and exception handling

- Lots of things can go wrong in computer programs
- For example, you may try to
  - Convert a string with non-digit characters to an integer
  - Call a non-existent function of a class
  - Import a non-existent module
  - Open a non-existent file
  - Modify an immutable object

These are examples of exceptions



Should our programs stop or handle these exceptions?

Exceptions

- Indication of a “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 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

```
1   def divide(num1, num2):
2       result = num1 / num2
3       return result
4
5   def foo(a, b):
6       D = divide(a, b)
7       print(D)
8
9   def main():
10      x = 10
11      y = 0
12      foo(x, y)
13
14  main()
```

```
In [1]: runfile('/Users/cigdem/Desktop/exception_example.py', wdir='/Users/
Traceback (most recent call last):

  File "/Users/cigdem/Desktop/exception_example.py", line 14, in <module>
    main()

  File "/Users/cigdem/Desktop/exception_example.py", line 12, in main
    foo(x, y)

  File "/Users/cigdem/Desktop/exception_example.py", line 6, in foo
    D = divide(a, b)

  File "/Users/cigdem/Desktop/exception_example.py", line 2, in divide
    result = num1 / num2

ZeroDivisionError: division by zero
```

Exceptions

- Many exceptions can be prevented by careful coding
  - For example, in order to avoid an error of converting a non-numeric string to an integer, you may first check if the string is made up of digits

- However, some exceptions cannot be avoided
- Hence, we need to handle them

Exception handling

- We must first “anticipate” an exception (including its type), then write code to “catch” it, and handle it

- The **try-except** statement

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

- The code in the except part is 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 an exception
  - If this is an exception specified in except clause
    - Handler, immediately following the except clause, executes
    - Continue program after the try-except statement
  - Other exceptions:
    - Program halts with a traceback error message
- If no exception is raised, handlers (statements in the except block) are skipped

Example

```
1   def divide(num1, num2):
2       result = num1 / num2
3       return result
4
5   def foo(a, b):
6       D = divide(a, b)
7       print(D)
8
9   def main():
10      x = 10
11      y = 0
12      foo(x, y)
13
14  main()
```

```
In [1]: runfile('/Users/cigdem/Desktop/exception_example.py', wdir='/Users/
Traceback (most recent call last):

  File "/Users/cigdem/Desktop/exception_example.py", line 14, in <module>
    main()

  File "/Users/cigdem/Desktop/exception_example.py", line 12, in main
    foo(x, y)

  File "/Users/cigdem/Desktop/exception_example.py", line 6, in foo
    D = divide(a, b)

  File "/Users/cigdem/Desktop/exception_example.py", line 2, in divide
    result = num1 / num2

ZeroDivisionError: division by zero
```

Example

```
def divide(num1, num2):
    try:
        result = num1 / num2
        return result
    except ZeroDivisionError:
        return 0

def foo(a, b):
    D = divide(a, b)
    print(D)

def main():
    x = 10
    y = 0
    foo(x, y)

main()
```

```
In [1]: runfil
wdir='/Users/c
0
```

Exception handling

- Code in the try block may throw more than one type of exception

- Need to write an except block (handler) for each type of the 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

Example

```
no = 100
while True:
    try:
        divisor = int(input('Enter an integer: '))
        result = no / divisor
        print('Result:',  format(result,'.2f'))
        break

    except ZeroDivisionError:
        print('Attempted to divide by zero')

    except ValueError:
        print('You must enter an integer')
```

```
In [1]: runfile('/Users/cigdem/De

Enter an integer: 124.5
You must enter an integer

Enter an integer: 0
Attempted to divide by zero

Enter an integer: comp 125
You must enter an integer

Enter an integer: 13
Result: 7.69
```

Can specify multiple except blocks

Compare it with the previous example

```
no = 100
while True:
    try:
        divisor = int(input('Enter an integer: '))
        result = no / divisor
        print('Result:',  format(result,'.2f'))
        break

    except (ZeroDivisionError, ValueError):
        print('Invalid divisor')
```

```
In [2]: runfile('/Users/cigdem/De

Enter an integer: 124.5
Invalid divisor

Enter an integer: 0
Invalid divisor

Enter an integer: comp 125
Invalid divisor

Enter an integer: 13
Result: 7.69
```

Some exceptions may be handled together, just write them as a tuple

Compare it with the previous examples

```
no = 100
while True:
    try:
        divisor = int(input('Enter an integer: '))
        result = no / divisor
        print('Result:',  format(result,'.2f'))
        break

    except ZeroDivisionError:
        print('Attempted to divide by zero')

    except:
        print('Other exception')
```

```
In [1]: runfile('/Users/cigdem/Des

Enter an integer: 124.5
Other exception

Enter an integer: 0
Attempted to divide by zero

Enter an integer: comp 125
Other exception

Enter an integer: 13
Result: 7.69
```

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

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)
```

Example

```
no = 100
while True:
    try:
        divisor = int(input('Enter an integer: '))
        result = no / divisor
        print('Result:',  format(result,'.2f'))
        break

    except ZeroDivisionError as err1:
        print(err1)

    except ValueError as err2:
        print(err2)
```

```
In [2]: runfile('/Users/cigdem/Desktop/example1.py'

Enter an integer: 124.5
invalid literal for int() with base 10: '124.5'

Enter an integer: 0
division by zero

Enter an integer: comp 125
invalid literal for int() with base 10: 'comp 125'

Enter an integer: 13
Result: 7.69
```

Using `else` with `try-except`

- `try/except` statement may include an optional `else` clause
  - It should be aligned with the try and except clauses and
  - It should appear after all the 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` block is skipped

Example

```
no = 100
while True:
    try:
        divisor = int(input('Enter an integer: '))
        result = no / divisor

    except ZeroDivisionError as err1:
        print(err1)

    except ValueError as err2:
        print(err2)

    else:
        print('Result:',  format(result,'.2f'))
        break
```

```
In [4]: runfile('/Users/cigdem/Desktop/example1.py

Enter an integer: 124.5
invalid literal for int() with base 10: '124.5'

Enter an integer: 0
division by zero

Enter an integer: comp 125
invalid literal for int() with base 10: 'comp 125'

Enter an integer: 13
Result: 7.69
```

`finally` clause

- `try/except` statement may include an optional `finally` clause
  - It should be aligned with the try and except clauses and
  - It should appear after all the except clauses

- The code within the finally block is executed whether an exception occurs or not

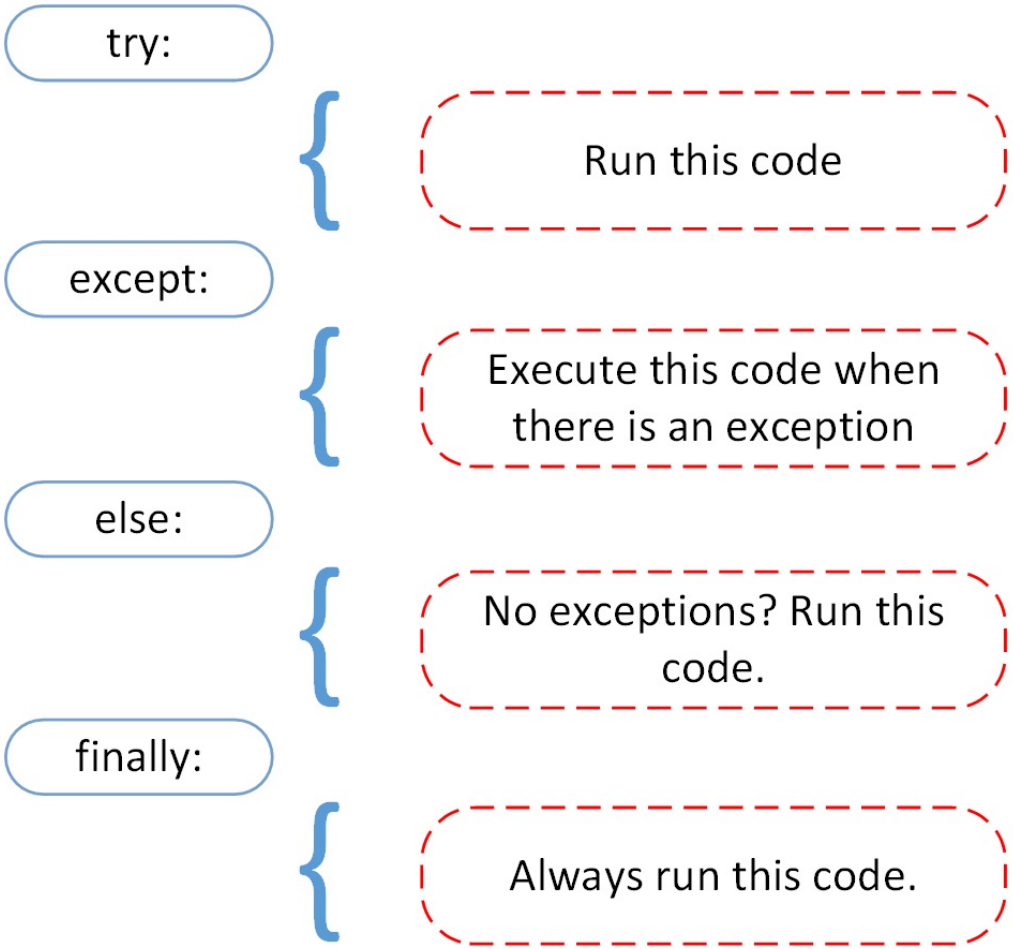
- Purpose is to perform cleanup

Example

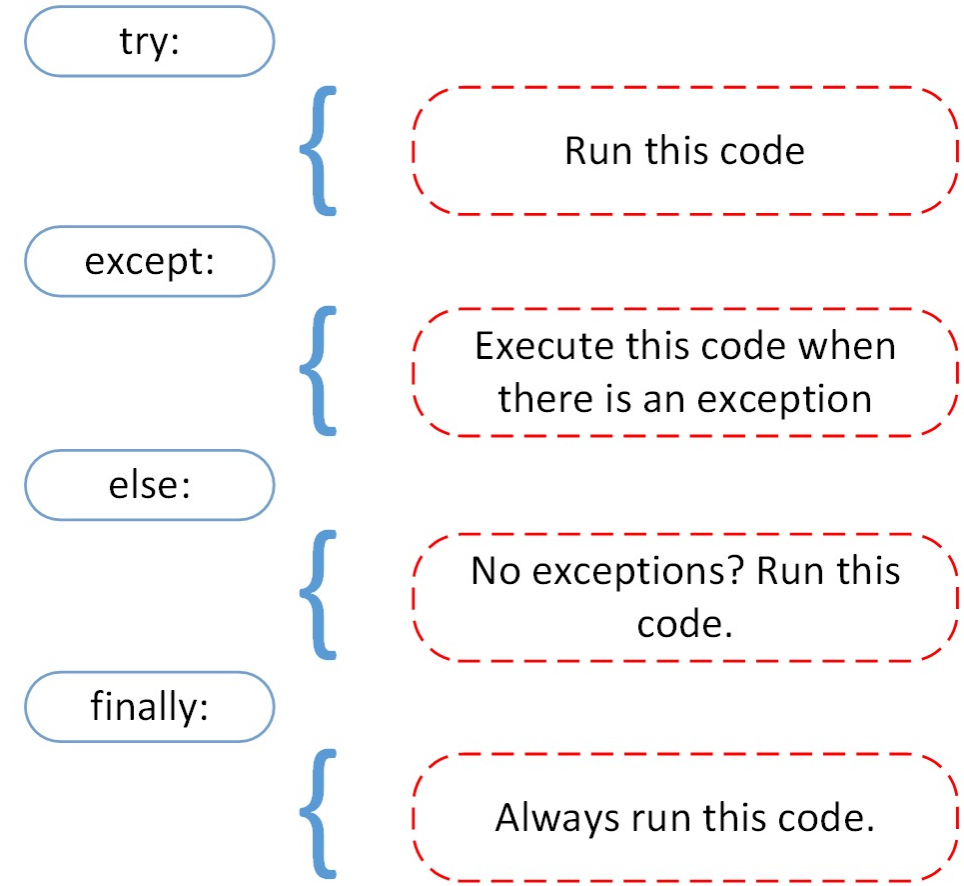
```
no = 100
trial_no = 0
while True:
    trial_no += 1
    try:
        divisor = int(input('Enter an integer: '))
        result = no / divisor

    except ZeroDivisionError as err1:
        print(err1)

    except ValueError as err2:
        print(err2)

    else:
        print('Result:',  format(result,'.2f'))
        break

    finally:
        print('Trial no:', trial_no)
```

```
In [6]: runfile('/Users/cigdem/Desktop/example1.py'

Enter an integer: 124.5
invalid literal for int() with base 10: '124.5'
Trial no: 1

Enter an integer: 0
division by zero
Trial no: 2

Enter an integer: comp 125
invalid literal for int() with base 10: 'comp 125'
Trial no: 3

Enter an integer: 13
Result: 7.69
Trial no: 4
```

Summary


try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.
finally:
Always run this code.

Summary



Source: Real Python

- Can have multiple except clauses
- Can handle multiple exceptions in a single except block
- Can get the exception object with as
- A lonely except catches “everything”

What happens if an exception is not handled?

- Two ways for exception to go unhandled
  - No except clause specifying the exception of the right type
  - Exception raised outside a try clause
- In both cases, exception will cause the program to halt (and a traceback)
- Python documentation provides information about exceptions that can be raised by different functions
  https://docs.python.org/3/library/exceptions.html#exception-hierarchy

Raising an exception

- You may want to raise your own exception, especially if you are writing a module/library

```
raise ExceptionType(message)
```

- A simple example

```
if row1 != row2:
    raise ValueError("Number of rows do not match")
```

- The parent of each exception type is the **`Exception`**. If unsure, just throw this

```
if a == 0:
    raise Exception("Number cannot be zero.")
```

Example

```
def square_root(n):
    if n < 0:
        raise Exception('Negative number')
    else:
        return n ** 0.5

def main():
    try:
        number = input('Enter a number: ')
        number = float(number)
        res = square_root(number)

    except ValueError:
        print('Enter a float')

    except:
        print('Enter a positive float')

    else:
        print('Result: ', res)

main()
```

```
In [1]: runfile('/Users/ci

Enter a number: comp125
Enter a float

In [2]: runfile('/Users/ci

Enter a number: -64
Enter a positive float

In [3]: runfile('/Users/ci

Enter a number: 65
Result:  8.06225774829855
```

Example

```
def square_root(n):
    if n < 0:
        raise Exception('Negative number')
    else:
        return n ** 0.5

def main():
    try:
        number = input('Enter a number: ')
        number = float(number)
        res = square_root(number)

    except Exception as err:
        print(err)

    else:
        print('Result: ', res)

main()
```

```
In [1]: runfile('/Users/cig

Enter a number: comp125
could not convert string to

In [2]: runfile('/Users/cig

Enter a number: -64
Negative number

In [3]: runfile('/Users/cig

Enter a number: 65
Result:  8.06225774829855
```