Files and Exception Handling

Massive Information & Knowledge Engineering

204113 Computer & Programming

Dr. Arnon Rungsawang
Dept. of computer engineering
Kasetsart University

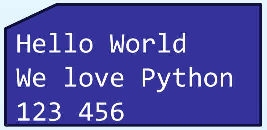
https://mike.cpe.ku.ac.th/204113

Version 2024

Python File Operation

File I/O and Operation

- A file is a container in computer storage devices used for storing data.
- When we want to read from or write to a file, we need to open it first.
- When we are done. It needs to be closed so that the resources that are tied with the file are freed.
- Hence, in Python, a file operation takes place in the following order:
  - Open a file
  - Read or write (perform operation)
  - Close the file
- From now, suppose that we have the following geek.txt file as an example:



```
Hello World
We love Python
123 456
```

Opening files

- In Python, we use the open() function to open files.

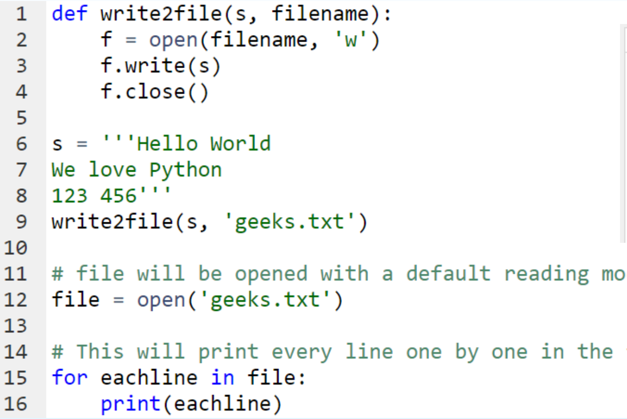
```
f = open(filename, mode)
```

where the mode could be:

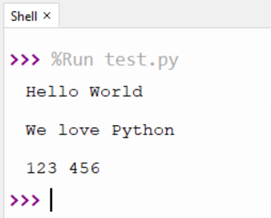
- 'r' open an existing file for a read operation.
- 'w' open an existing file for a write operation. If the file already contains some data, then it will be overridden but if the file is not present then it creates the file as well.
- 'a' open an existing file for append operation. It won't override existing data.
- 'r+' To read and write data into the file. The previous data in the file will be overridden.
- 'w+' To write and read data. It will override existing data.
- 'a+' To append and read data from the file. It won't override existing data.

Read from a file

- The following open() function will open the file in the read mode and the for loop will print each line presented in the file.



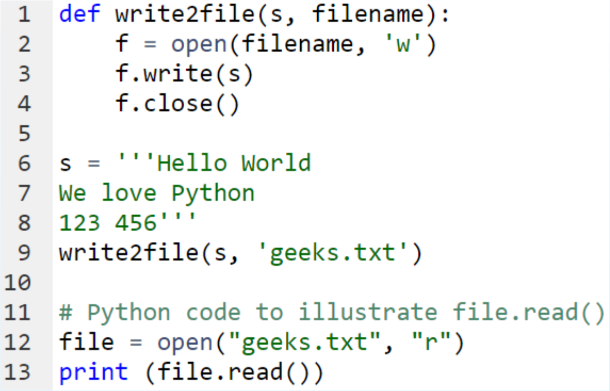
```python
def write2file(s, filename):
    f = open(filename, 'w')
    f.write(s)
    f.close()

s = '''Hello World
We love Python
123 456'''
write2file(s, 'geeks.txt')

# file will be opened with a default reading mode.
file = open('geeks.txt')

# This will print every line one by one in the file
for eachline in file:
    print(eachline)
```



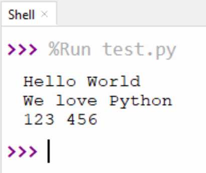
```
>>> %Run test.py
 Hello World

 We love Python

 123 456
>>> |
```

Read from a file (2)

- In this example, we will use the file.read() method to extract a string that contains all characters in the file.



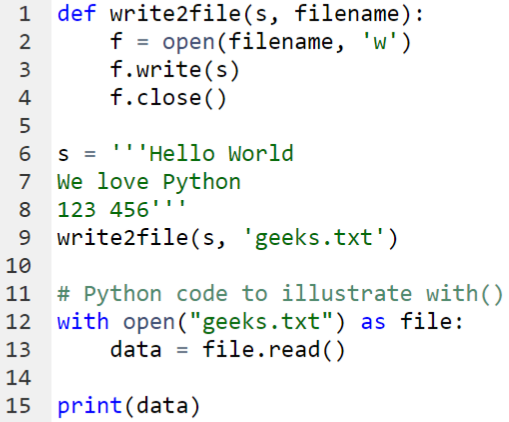
```python
def write2file(s, filename):
    f = open(filename, 'w')
    f.write(s)
    f.close()

s = '''Hello World
We love Python
123 456'''
write2file(s, 'geeks.txt')

# Python code to illustrate file.read()
file = open("geeks.txt", "r")
print (file.read())
```



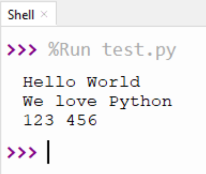
```
>>> %Run test.py
 Hello World
 We love Python
 123 456
>>> |
```

Read from a file (3) – with statement

- In this example, we will see how we read a file using the with statement.



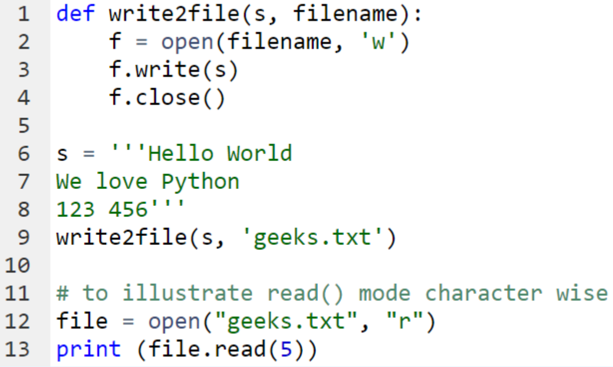
```python
def write2file(s, filename):
    f = open(filename, 'w')
    f.write(s)
    f.close()

s = '''Hello World
We love Python
123 456'''
write2file(s, 'geeks.txt')

# Python code to illustrate with()
with open("geeks.txt") as file:
    data = file.read()

print(data)
```



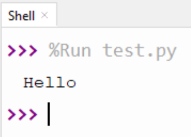
```
>>> %Run test.py
 Hello World
 We love Python
 123 456
>>> |
```

Read from a file (4)

- Another way to read a file is to call a certain number of characters like in the following code. The interpreter will read the first five characters of stored data and return it as a string.



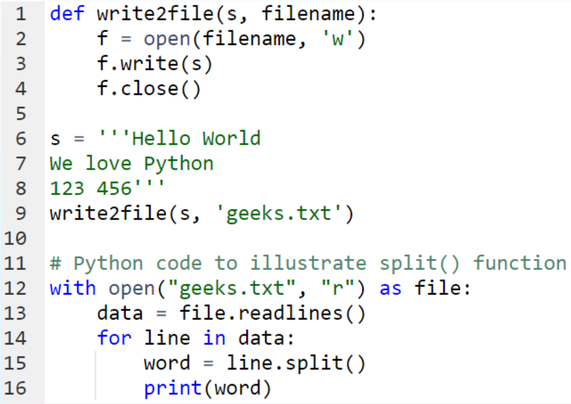
```python
def write2file(s, filename):
    f = open(filename, 'w')
    f.write(s)
    f.close()

s = '''Hello World
We love Python
123 456'''
write2file(s, 'geeks.txt')

# to illustrate read() mode character wise
file = open("geeks.txt", "r")
print (file.read(5))
```



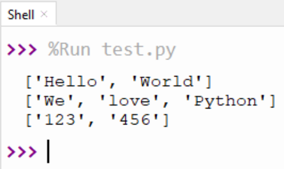
```
>>> %Run test.py
 Hello
>>> |
```

Read from a file (5)

- We can also split lines while reading files in Python. The `split()` function splits the variable when space is encountered. We can also split using any characters as we wish.



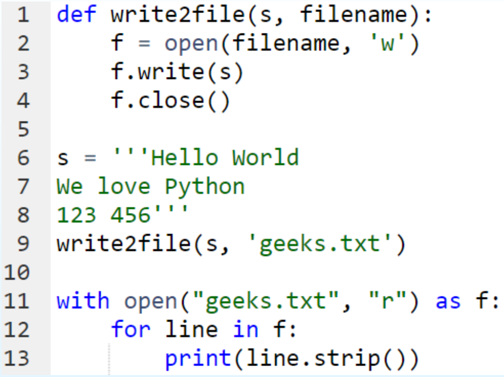
```python
def write2file(s, filename):
    f = open(filename, 'w')
    f.write(s)
    f.close()

s = '''Hello World
We love Python
123 456'''
write2file(s, 'geeks.txt')

# Python code to illustrate split() function
with open("geeks.txt", "r") as file:
    data = file.readlines()
    for line in data:
        word = line.split()
        print(word)
```



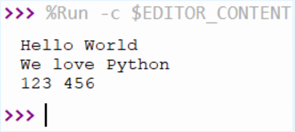
```
>>> %Run test.py
 ['Hello', 'World']
 ['We', 'love', 'Python']
 ['123', '456']
>>> |
```

Read from a file (6)

- The following code show how to read file content using a `for` loop.



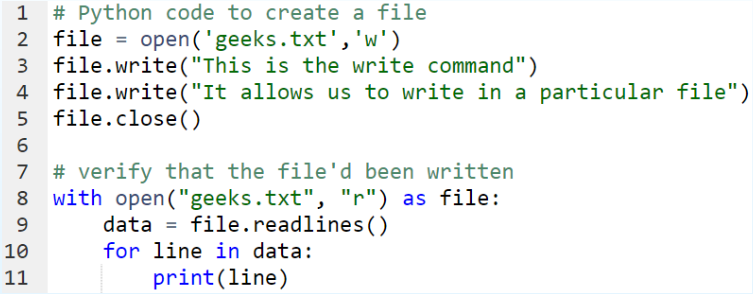
```python
def write2file(s, filename):
    f = open(filename, 'w')
    f.write(s)
    f.close()

s = '''Hello World
We love Python
123 456'''
write2file(s, 'geeks.txt')

with open("geeks.txt", "r") as f:
    for line in f:
        print(line.strip())
```



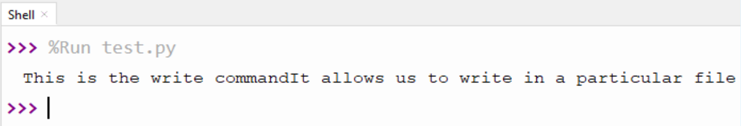
```
>>> %Run -c $EDITOR_CONTENT
 Hello World
 We love Python
 123 456
>>> |
```

Write to a file

- In this example, we will see how the `write()` function is used to write in a file.
- The `close()` command terminates all the resources in use and frees the system of this particular program.



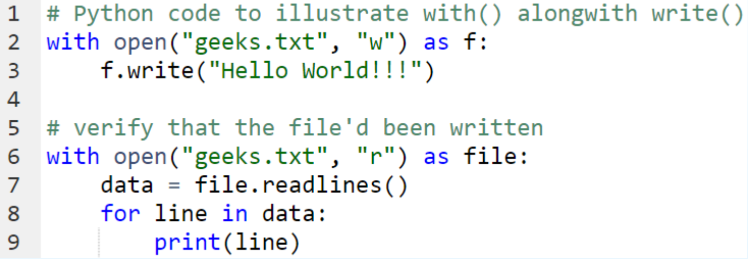
```python
# Python code to create a file
file = open('geeks.txt','w')
file.write("This is the write command")
file.write("It allows us to write in a particular file")
file.close()

# verify that the file'd been written
with open("geeks.txt", "r") as file:
    data = file.readlines()
    for line in data:
        print(line)
```



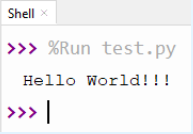
```
>>> %Run test.py
 This is the write commandIt allows us to write in a particular file
>>> |
```

Write to a file (2)

- We can also use the written statement along with the `with` statement.

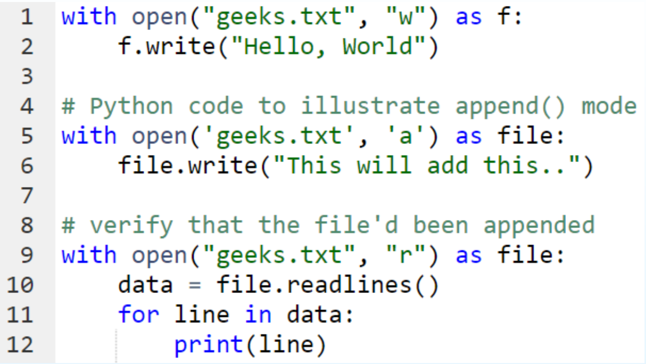


```python
# Python code to illustrate with() alongwith write()
with open("geeks.txt", "w") as f:
    f.write("Hello World!!!")

# verify that the file'd been written
with open("geeks.txt", "r") as file:
    data = file.readlines()
    for line in data:
        print(line)
```

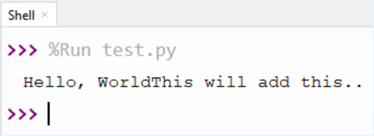


```
>>> %Run test.py
 Hello World!!!
>>> |
```

Append to an existing file



```
with open("geeks.txt", "w") as f:
    f.write("Hello, World")

# Python code to illustrate append() mode
with open('geeks.txt', 'a') as file:
    file.write("This will add this..")

# verify that the file'd been appended
with open("geeks.txt", "r") as file:
    data = file.readlines()
    for line in data:
        print(line)
```



```
>>> %Run test.py
 Hello, WorldThis will add this..
>>> 
```

Python Exception

Python Exceptions

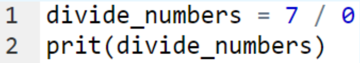
- An exception is an unexpected event that occurs during program execution.

```
divide_by_zero = 5 / 0
```

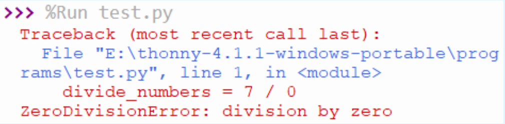
The above code causes an exception as it is not possible to divide a number by 0.

Python Logical Error (Exception)

- Errors that occur at runtime (after passing the syntax test) are called exceptions or logical errors.
- For instance, they occur when we
  - try to open a file (for reading) that does not exist (`FileNotFoundError`)
  - try to divide a number by zero (`ZeroDivisionError`)
  - try to import a module that does not exist (`ImportError`) and so on.
- Whenever these types of runtime errors occur, Python creates an exception object.
- If not handled properly, it prints a traceback to that error along with some details about why that error occurred.



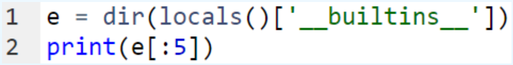
```
divide_numbers = 7 / 0
prit(divide_numbers)
```



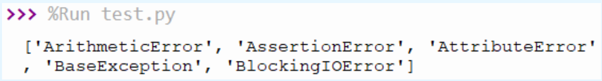
```
>>> %Run test.py
 Traceback (most recent call last):
   File "E:\thonny-4.1.1-windows-portable\prog
 rams\test.py", line 1, in <module>
     divide_numbers = 7 / 0
 ZeroDivisionError: division by zero
```

Python built-in Exception

- Illegal operations can raise exceptions.
- There are plenty of built-in exceptions in Python that are raised when corresponding errors occur.
- We can view all the built-in exceptions using the built-in `locals()` function.
  - Here, `locals()['__builtins__']` will return a module of built-in exceptions, functions, and attributes.
  - `dir` allows us to list those attributes as strings.



```python
e = dir(locals()['__builtins__'])
print(e[:5])
```



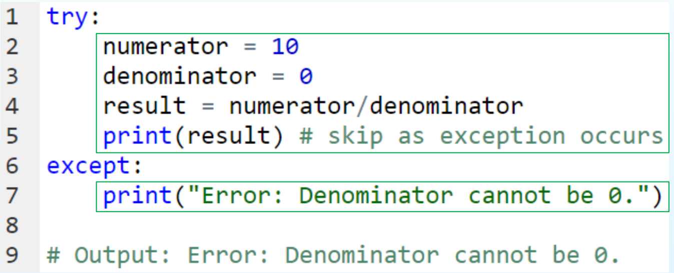
```
>>> %Run test.py
 ['ArithmeticError', 'AssertionError', 'AttributeError'
 , 'BaseException', 'BlockingIOError']
```

Common built-in Exceptions

Exception	Cause of Error
`EOFError`	Raised when the input() function hits the end-of-file condition
`ImportError`	Raised when the imported module is not found
`IndexError`	Raised when the index of a sequence is out of range
`KeyError`	Raised when a key is not found in a dictionary
`NameError`	Raised when a variable is not found in local or global scope
`IndentationError`	Raised when there is incorrect indentation !
`TypeError`	Raised when a function or operation is applied to an object of incorrect type
`ValueError`	Raised when a function gets and argument of correct type but improper value
`ZeroDivisionError`	Raised when the second operand of division or modulo operation is zero

Exception Handling try…except

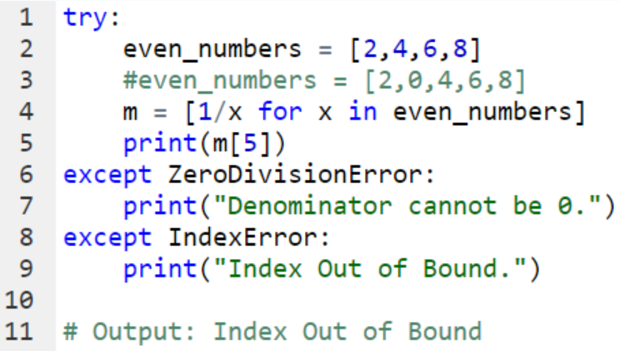
- The `try…except` block is used to handle exceptions in Python.
- We place the code that might generate an exception inside the `try` block.
- Every `try` block is followed by an `except` block.



```python
try:
    numerator = 10
    denominator = 0
    result = numerator/denominator
    print(result) # skip as exception occurs
except:
    print("Error: Denominator cannot be 0.")

# Output: Error: Denominator cannot be 0.
```

Catching specific Exceptions

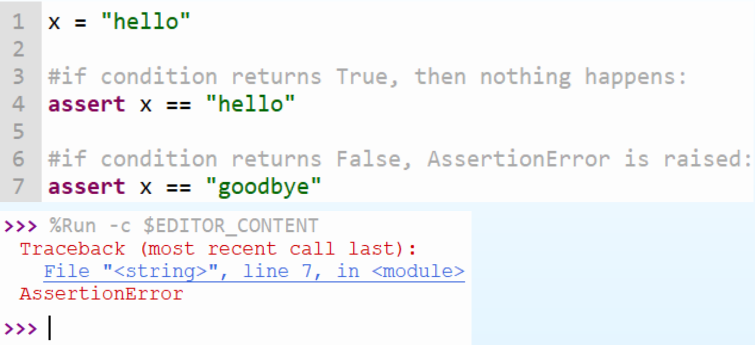
- For each `try` block, there can be one or more `except` block.
- Multiple `except` block allow us to handle each exception differently.



```python
try:
    even_numbers = [2,4,6,8]
    #even_numbers = [2,0,4,6,8]
    m = [1/x for x in even_numbers]
    print(m[5])
except ZeroDivisionError:
    print("Denominator cannot be 0.")
except IndexError:
    print("Index Out of Bound.")

# Output: Index Out of Bound
```

assert statement

- Test whether a condition return True.

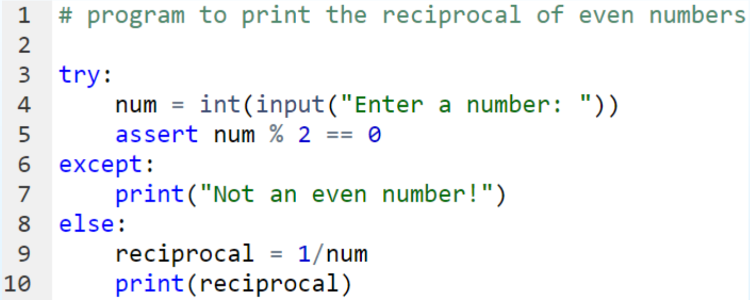


```
1 x = "hello"
2
3 #if condition returns True, then nothing happens:
4 assert x == "hello"
5
6 #if condition returns False, AssertionError is raised:
7 assert x == "goodbye"
```

```
>>> %Run -c $EDITOR_CONTENT
 Traceback (most recent call last):
   File "<string>", line 7, in <module>
 AssertionError
>>> |
```

try…except…else

- In some situations, we might want to run a certain block of code if the code block inside try runs without any errors.
- For these cases, we can use the optional else keyword with the try statement.

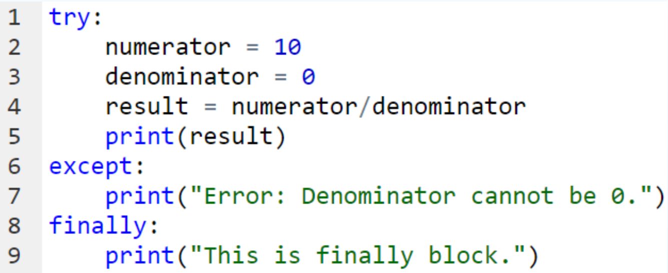


```
# program to print the reciprocal of even numbers

try:
    num = int(input("Enter a number: "))
    assert num % 2 == 0
except:
    print("Not an even number!")
else:
    reciprocal = 1/num
    print(reciprocal)
```

- However, if we enter 0 as input, we get ZeroDivisionError as the code block inside else is not handled correctly by preceding except.

try…finally

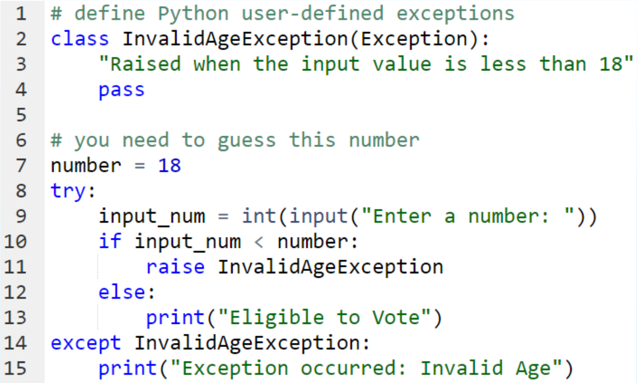
- In Python, the finally block is always executed no matter whether there is an exception or not.
- The finally block is optional. And, for each try block, there can be only one finally block.



```
try:
    numerator = 10
    denominator = 0
    result = numerator/denominator
    print(result)
except:
    print("Error: Denominator cannot be 0.")
finally:
    print("This is finally block.")
```

Defining custom exceptions

- We can define custom exceptions by creating a new class that is derived from the built-in Exception class.

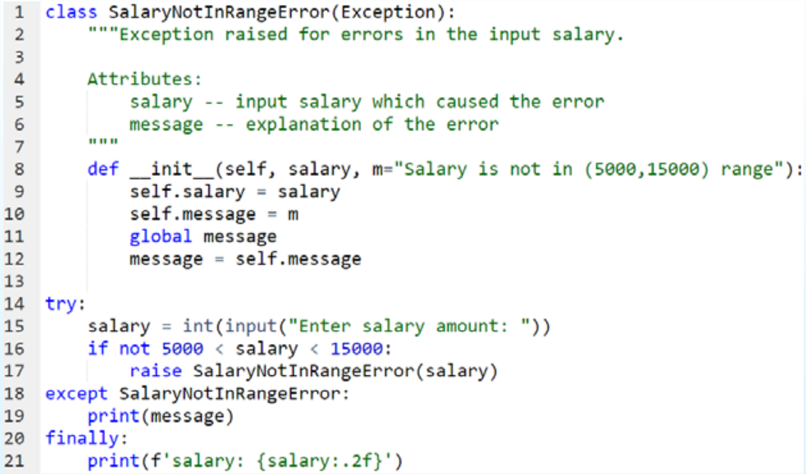


```
# define Python user-defined exceptions
class InvalidAgeException(Exception):
    "Raised when the input value is less than 18"
    pass

# you need to guess this number
number = 18
try:
    input_num = int(input("Enter a number: "))
    if input_num < number:
        raise InvalidAgeException
    else:
        print("Eligible to Vote")
except InvalidAgeException:
    print("Exception occurred: Invalid Age")
```

- When an exception occurs, the rest of the code inside the try block is skipped.
- The except block catches the user-defined InvalidAgeException exception and statements inside the except block are executed.

Customizing exception classes

- We can further customize this class to accept other arguments as per our needs.



```python
class SalaryNotInRangeError(Exception):
    """Exception raised for errors in the input salary.

    Attributes:
        salary -- input salary which caused the error
        message -- explanation of the error
    """
    def __init__(self, salary, m="Salary is not in (5000,15000) range"):
        self.salary = salary
        self.message = m
        global message
        message = self.message

try:
    salary = int(input("Enter salary amount: "))
    if not 5000 < salary < 15000:
        raise SalaryNotInRangeError(salary)
except SalaryNotInRangeError:
    print(message)
finally:
    print(f'salary: {salary:.2f}')
```

Python Error and Exception

- Errors represent conditions such as compilation error, syntax error, error in the logical part of the code, library incompatibility, infinite recursion, etc.
- Errors are usually beyond the control of the programmer, and we should not try to handle errors.
- Exceptions can be caught and handled by the program.

Problem Solving Samples

Task: Score Ranking

- Read a file containing a list of *scores*.
- Then sort the scores from highest to lowest and print out the ranking.

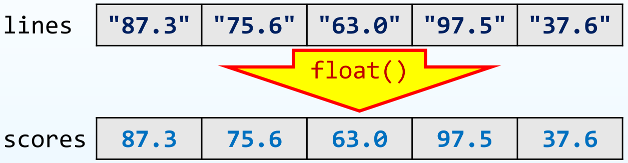
```
Enter score file: scores.txt
Rank #1: 97.5
Rank #2: 87.3
Rank #3: 75.6
Rank #4: 63.0
Rank #5: 37.6
```

scores.txt

```
87.3
75.6
63.0
97.5
37.6
```

Score Ranking – Ideas

- Scores must be read as *a list of numbers*, not strings.
- Each string member must get converted into a number.



- Straightforward code with a for loop:

```
:
lines = open(filename).read().splitlines()
scores = []
for x in lines:
    scores.append(float(x))
:
```

Score Ranking – Ideas

- With a list comprehension, the code

```
scores = []
for x in lines:
    scores.append(float(x))
```

can be replaced by a much more concise statement:

```
scores = [float(x) for x in lines]
```

Score Ranking – Program

```
filename = input("Enter score file: ")
lines = open(filename).read().splitlines()
scores = [float(x) for x in lines]
scores.sort(reverse=True)
for i in range(len(scores)):
    print(f"Rank #{i+1}: {scores[i]}")
```

Sort the scores from highest to lowest

```
Enter score file: scores.txt
Rank #1: 97.5
Rank #2: 87.3
Rank #3: 75.6
Rank #4: 63.0
Rank #5: 37.6
```

scores.txt

```
87.3
75.6
63.0
97.5
37.6
```

Caveats – Empty Lines in File

- Empty lines in the input file will break the program.

```
Enter score file: scores.txt
Traceback (most recent call last):
  File "score-rank.py", line 3, in <module>
    scores = [float(x) for x in lines]
  File "score-rank.py", line 3, in <listcomp>
    scores = [float(x) for x in lines]
ValueError: could not convert string to float:
```

scores.txt

```
87.3
75.6

63.0
97.5
37.6

```

empty line

empty line

- We must filter out those empty lines before converting them to floats.

Score Ranking – Revised Program

- This version skips empty lines in the input file.

```
filename = input("Enter score file: ")
lines = open(filename).read().splitlines()
scores = [float(x) for x in lines if x != ""]
scores.sort(reverse=True)
for i in range(len(scores)):
    print(f"Rank #{i+1}: {scores[i]}")
```

This condition helps skip empty lines

```
Enter score file: scores.txt
Rank #1: 97.5
Rank #2: 87.3
Rank #3: 75.6
Rank #4: 63.0
Rank #5: 37.6
```

scores.txt

```
87.3
75.6

63.0
97.5
37.6
```

CSV Files

- **C**omma-**S**eparated **V**alues
- Commonly used to store tabular data as a text file.
  - Each line is a row.
  - Columns in each line (row) are separated by commas.

Subject	Credits	Grade
01175112	1	B+
01204111	3	A
01417167	3	B

rows

columns

grades.txt

```
01175112,1,B+
01204111,3,A
01417167,3,B
```

- CSV files can be opened directly in Microsoft Excel.

Task: GPA Calculator

- Read a CSV file containing a list of *subject codes*, their *credits*, and the *grades* received.
- Then display *grade summary*, *total credits*, and *GPA.*

```
Enter grade data file: grades.txt
----------------------------------
  Subject   Credits  Grade  Point
----------------------------------
  01175112     1        B+     3.5
  01204111     3        A      4.0
  01355112     3        C+     2.5
  01417167     3        B      3.0
----------------------------------
Total credits = 10
GPA = 3.20
```

grades.txt

```
01175112,1,B+
01204111,3,A
01355112,3,C+
01417167,3,B
```

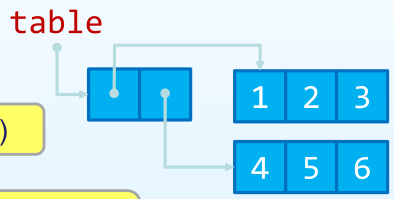
GPA Calculator – Ideas

- How to store tabular data in Python?
  - A table is a list of rows; each row is a list of columns.
- We need a ***list of lists***
  - also known as a ***nested list***

```
>>> table = [[1,2,3],[4,5,6]]
>>> len(table)
2
>>> table[1]
[4, 5, 6]
>>> table[1][2]
6
```

Access row#1 (2nd row)

Access column#2 (3rd column) in row#1 (2nd row)



GPA Calculator – Steps

- Divide the whole task into three major steps
  - **Step 1:** read grade table data from file as a nested list
  - **Step 2:** display the grade table
  - **Step 3:** calculate total credits and GPA

Breaking Lines into Columns

- Python provides `str.split()` method.

```
>>> line = "01204111,3,A"
>>> line.split(",")
['01204111', '3', 'A']
```

grades.txt

```
01175112,1,B+
01204111,3,A
01355112,3,C+
01417167,3,B
```

- Let us try using it inside a list comprehension.

```
>>> lines = open("grades.txt").read().splitlines()
>>> lines
['01175112,1,B+', '01204111,3,A', '01355112,3,C+', '01417167,3,B']
>>> table = [x.split(",") for x in lines]
>>> table
[['01175112', '1', 'B+'], ['01204111', '3', 'A'], ['01355112',
'3', 'C+'], ['01417167', '3', 'B']]
```

We now got a nested list!

GPA Calculator – Steps

Step 1 - read grade table from file as a nested list

- We will define `read_table()` function as follows:

```
def read_table(filename):
    lines = open(filename).read().splitlines()
    table = [x.split(",") for x in lines if x != ""]
    return table
```

grades.txt

```
01175112,1,B+
01204111,3,A
01355112,3,C+
01417167,3,B
```

- Let's test it

```
>>> read_table("grades.txt")
[['01175112', '1', 'B+'], ['01204111', '3', 'A'], ['01355112',
'3', 'C+'], ['01417167', '3', 'B']]
```

GPA Calculator – Steps

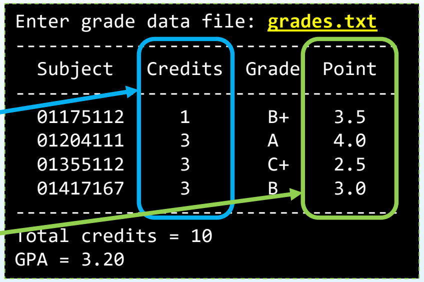
- The resulting table is not complete

```
>>> read_table("grades.txt")
[['01175112', '1', 'B+'], ['01204111', '3', 'A'],
['01355112', '3', 'C+'], ['01417167', '3', 'B']]
```

grades.txt

```
01175112,1,B+
01204111,3,A
01355112,3,C+
01417167,3,B
```

- Output on the right is what we expect to get in the end
  - The credits column should store integers, not strings, for later calculation
  - The point column is still missing



```
Enter grade data file: grades.txt
--------------------------------------
  Subject   Credits  Grade  Point
--------------------------------------
  01175112     1       B+     3.5
  01204111     3       A      4.0
  01355112     3       C+     2.5
  01417167     3       B      3.0
--------------------------------------
Total credits = 10
GPA = 3.20
```

GPA Calculator – Steps

- We will traverse the table list to perform adjustment on each row.
  - We also define grade_point() function to map a grade to a point.

```
def read_table(filename):
    lines = open(filename).read().splitlines()
    table = [x.split(",") for x in lines if x != ""]
    for row in table:
        # convert credits to integers
        row[1] = int(row[1])
        # add a new column for grade point
        row.append(grade_point(row[2]))
    return table
```

```
>>> table = read_table("grades.txt")
>>> table
[['01175112', 1, 'B+', 3.5], ['01204111', 3,
'A', 4.0], ['01355112', 3, 'C+', 2.5],
['01417167', 3, 'B', 3.0]]
```

```
def grade_point(grade):
    if grade == "A":
        return 4.0
    elif grade == "B+":
        return 3.5
    elif grade == "B":
        return 3.0
    elif grade == "C+":
        return 2.5
    elif grade == "C":
        return 2.0
    elif grade == "D+":
        return 1.5
    elif grade == "D":
        return 1.0
    elif grade == "F":
        return 0.0
```

GPA Calculator – Steps

Step 2 - display the grade table

- Traverse the table list and print out each row.

```
def print_table(table):
    print("-----------------------------------")
    print("  Subject   Credits  Grade  Point")
    print("-----------------------------------")
    for row in table:
        print(f"  {row[0]:8} {row[1]:5}      {row[2]:<5} {row[3]:.1f}")
    print("-----------------------------------")
```

```
>>> print_table(table)   # table from previous step
-----------------------------------
  Subject   Credits  Grade  Point
-----------------------------------
  01175112    1        B+    3.5
  01204111    3        A     4.0
  01355112    3        C+    2.5
  01417167    3        B     3.0
-----------------------------------
```

Not so difficult, but a lot of tweaking to get a nice-looking table

GPA Calculator – Steps

Step 3 - calculate total credits and GPA

- Total of credits is computed from the summation of column#1 in all rows.

```
total_credits = sum([row[1] for row in table])
```

```
>>> [row[1] for row in table]
[1, 3, 3, 3]
```

```
>>> table
[['01175112', 1, 'B+', 3.5],
['01204111', 3, 'A', 4.0],
['01355112', 3, 'C+', 2.5],
['01417167', 3, 'B', 3.0]]
```

GPA Calculator – Steps

Step 3 - calculate total credits and GPA

- GPA is computed from the summation of credits*point of all subjects
  - credits → column#1, point → column#3

```
>>> [row[1]*row[3] for row in table]
[3.5, 12.0, 7.5, 9.0]
```

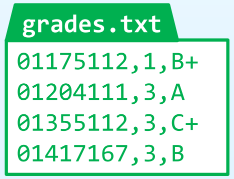
```
>>> table
[['01175112', 1, 'B+', 3.5],
['01204111', 3, 'A', 4.0],
['01355112', 3, 'C+', 2.5],
['01417167', 3, 'B', 3.0]]
```

```
sum_credits_point = sum([row[1]*row[3] for row in table])
gpa = sum_credits_point/total_credits
```

GPA Calculator – Main Program

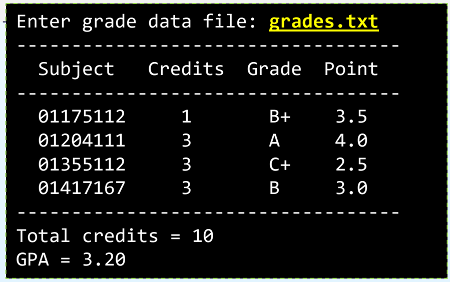
- `read_table()` and `print_table()` are not shown.

```
filename = input("Enter grade data file: ")
table = read_table(filename)
print_table(table)
total_credits = sum([row[1] for row in table])
sum_credits_point = sum([row[1]*row[3] for row in table])
gpa = sum_credits_point/total_credits
print(f"Total credits = {total_credits}")
print(f"GPA = {gpa:.2f}")
```



grades.txt

```
01175112,1,B+
01204111,3,A
01355112,3,C+
01417167,3,B
```



```
Enter grade data file: grades.txt
-----------------------------------
  Subject   Credits  Grade  Point
-----------------------------------
  01175112     1       B+    3.5
  01204111     3       A     4.0
  01355112     3       C+    2.5
  01417167     3       B     3.0
-----------------------------------
Total credits = 10
GPA = 3.20
```

To be continue..

つづく