PYTHON

FILES AND EXCEPTIONS

NOR HAMIZAH MISWAN

READING FROM A FILE

- An incredible amount of data is available in text files.
- Text files can contain weather data, traffic data, socioeconomic data, literary works, and more.
- Reading from a file is particularly useful in data analysis applications, but it's also applicable to any situation in which you want to analyze or modify information stored in a file.
- When you want to work with the information in a text file, the first step is to read the file into memory.

READING FROM A FILE

- To do any work with a file, even just printing its contents, you first need to open the file to access it.
- The open() function needs one argument: the name of the file you want to open.

Try:

```
with open('pi_digits.txt') as file_object:
    contents = file_object.read()
    print(contents)
```

- The open() function returns an object representing the file. Python stores this object in file_object, which we'll work with later in the program.

READING FROM A FILE

- If we want to looks file in other directory, set the file_path. How?

  file_path = 'C:/Users/Hamizah/Downloads/pi_digits.txt'

- Then use: with open(file_path) as file_object:

Try:

```
with open(file_path) as file_object:
    contents = file_object.read()
    print(contents)
```

- The open() function returns an object representing the file. Python stores this object in file_object, which we'll work with later in the program.

READING FROM A FILE

- Or directly paste the directory in the open () function:

Try:

```
with open('C:/Users/Hamizah/Downloads/pi_digits.txt') as file_object:
    contents = file_object.read()
    print(contents)
```

READING FROM A FILE

- Sometime we want to examine each line of the file. Especially when looking for certain information in the file, or might want to modify the text in the file in some way.
- You can use a for loop on the file object to examine each line from a file one at a time:

Try:

```
filename = 'pi_digits.txt'

with open(filename) as file_object:
    for line in file_object:  #use for loop
        print(line)
```

```
 #or use rstrip() function eliminate extra
blank lines
print(line.rstrip())
```

WRITING TO A FILE

- One of the simplest ways to save data is to write it to a file.
- When you write text to a file, the output will still be available after you close the terminal containing your program's output.
- Some ways to write a file:
  - Writing to an empty file
  - Writing multiple lines
  - Appending to a file

WRITING TO A FILE

- To write text to a file, you need to call open() with a second argument telling Python that you want to write to the file.
- The call to open() in this example has two arguments:
  - ➢The first argument is still the name of the file we want to open.
  - ➢The second argument, 'w', tells Python that we want to open the file in write mode.
- The open() function automatically creates the file you're writing to if it doesn't already exist.
- However, be careful opening a file in write mode ('w') because if the file does exist, Python will erase the file before returning the file object.

WRITING TO A FILE

Try:

```
filename = 'programming.txt'

with open(filename, 'w') as file_object:
    file_object.write("I love programming.")
```

- It will save the object as the filename and store it in the same directory.

WRITING TO A FILE

- The write() function doesn’t add any newlines to the text you write.
- To write multiple lines, how?

Try:

```
filename = 'programming.txt'

with open(filename, 'w') as file_object:
    file_object.write("I love programming.\n")
    file_object.write("I love creating new games.\n")
```

WRITING TO A FILE

- If you want to add content to a file instead of writing over existing content, you can open the file in append mode. How?

Try:

```
filename = 'programming.txt'

with open(filename, 'a') as file_object:
    file_object.write("I also love finding meaning in large datasets.\n")
    file_object.write("I love creating apps that can run in a browser.\n")
```

- When you open a file in append mode, Python doesn’t erase the file before returning the file object. Any lines you write to the file will be added at the end of the file.

EXCEPTIONS

- Special objects Python creates to manage errors that arise while a program is running.
- Whenever an error occurs that makes Python unsure what to do next, it creates an exception object.
- Exceptions are handled with try-except blocks.
- A try-except block asks Python to do something, but it also tells Python what to do if an exception is raised.
- When you use try-except blocks, your programs will continue running even if things start to go wrong.

EXCEPTIONS

- Common error that causes Python to raise an exception:
  - ➢ZeroDivisionError
  - ➢FileNotFoundError
- Hence, we may write a try-except or try-except-else to handle these errors.

EXCEPTIONS

- Handling the ZeroDivisionError Exception:
  - You may try this:

    ```
    print(5/0)
    ```

  - The above solution can be solved using try-except block:

```
try:
    print(5/0) #the code
except ZeroDivisionError:
    print("You can't divide by zero!") #print the error
```

EXCEPTIONS

- Handling the ZeroDivisionError Exception:
  - You may try this:

```
print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")

while True:
    first_number = input("\nFirst number: ")
    if first_number == 'q':
        break
    second_number = input("Second number: ")
    if second_number == 'q':
        break
    answer = int(first_number) / int(second_number)
    print(answer)
```

- The above solution can be solved using try-except-else block:

```
print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")
while True:
    first_number = input("\nFirst number: ")
    if first_number == 'q':
        break
    second_number = input("Second number: ")
    try:
        answer = int(first_number) / int(second_number) #the
code
    except ZeroDivisionError:
        print("You can't divide by 0!") #print the error
    else:
        print(answer)
```

EXCEPTIONS

- Handling the FileNotFoundError Exception:
  - ➢ You may try this:

```
filename = 'alice.txt'

with open(filename) as f_obj:
    contents = f_obj.read()
```

➢ The above solution can be solved using try-except block:

```
filename = 'alice.txt'

try:
    with open(filename) as f_obj: #the code for
opening file
        contents = f_obj.read()
except FileNotFoundError:
    msg = "Sorry, the file " + filename + " does not
exist."
    print(msg) #print the error
```

EXCEPTIONS

- In the previous example, we informed our users that one of the files was unavailable.
- But you don't need to report every exception you catch.
- Sometimes you'll want the program to fail silently when an exception occurs and continue on as if nothing happened. How?
  - Python has a pass statement that tells it to do nothing in a block

EXCEPTIONS

Try to handle error silently:

```
print("Give me two numbers, and I'll divide them.")
print("Enter 'q' to quit.")

while True:
    first_number = input("\nFirst number: ")
    if first_number == 'q':
        break
    second_number = input("Second number: ")
    try:
        answer = int(first_number) / int(second_number)
    except ZeroDivisionError:
        pass
    else:
        print(answer)
```

STORING DATA

- When users close a program, you'll almost always want to save the information they entered.
- A simple way to do this involves storing your data using the json (JavaScript Object Notation) module:
  - ➢The json module allows you to dump simple Python data structures into a file and load the data from that file the next time the program runs.
  - ➢The json module can be used to share data between different Python programs.
  - ➢The JSON data format is not specific to Python, so you can share data you store in the JSON format with people who work in many other programming

STORING DATA

- Let’s write a short program that stores a set of numbers and another program that reads these numbers back into memory.
- The first program will use json.dump() to store the set of numbers.
  - ➢The json.dump() function takes two arguments: a piece of data to store and a file object it can use to store the data.
  - ➢Example: store a list of numbers

```
import json

numbers = [2, 3, 5, 7, 11, 13]
filename = 'numbers.json'
with open(filename, 'w') as f_obj:
    json.dump(numbers, f_obj)
```

- open the file in write mode allows json to write the data to the file.
- At w we use the json.dump() function to store the list numbers in the file numbers.json.

STORING DATA

- Now we'll write a program that uses json.load() to read the list back into memory.

```
import json

filename = 'numbers.json'
with open(filename) as f_obj:
    numbers = json.load(f_obj)

print(numbers)
```

➢ The json.load() function to load the information stored in numbers.json, and we store it in the variable numbers.