FILES
INPUT/OUTPUT

Insight Builder

PIPES & BUCKETS

PROBLEM SOLVED BY FILES I/O

1 UNDERSTAND FILES

- Files are store data in a "Encoded" format
- Has properties like location, size, num of lines, data so considered as an Object

2 WHAT IS THE PROBLEM?

Data has to move into and out of Python script in huge volume, velocity and variety

3 HOW IT SOLVES THEM

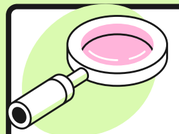
- Concept of Templates in Python allows to modify how a file is read based on Volume, Velocity and Variety

EXAMPLE

FILE OBJECTS

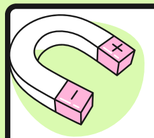
- regular_file = open('file.txt')
- pandas_csv = pd.read_csv('data.csv')

VALUE



- 'I am a file.'
- <Table of rows & cols>

TEMPLATE



- **String**
- **DataFrame**

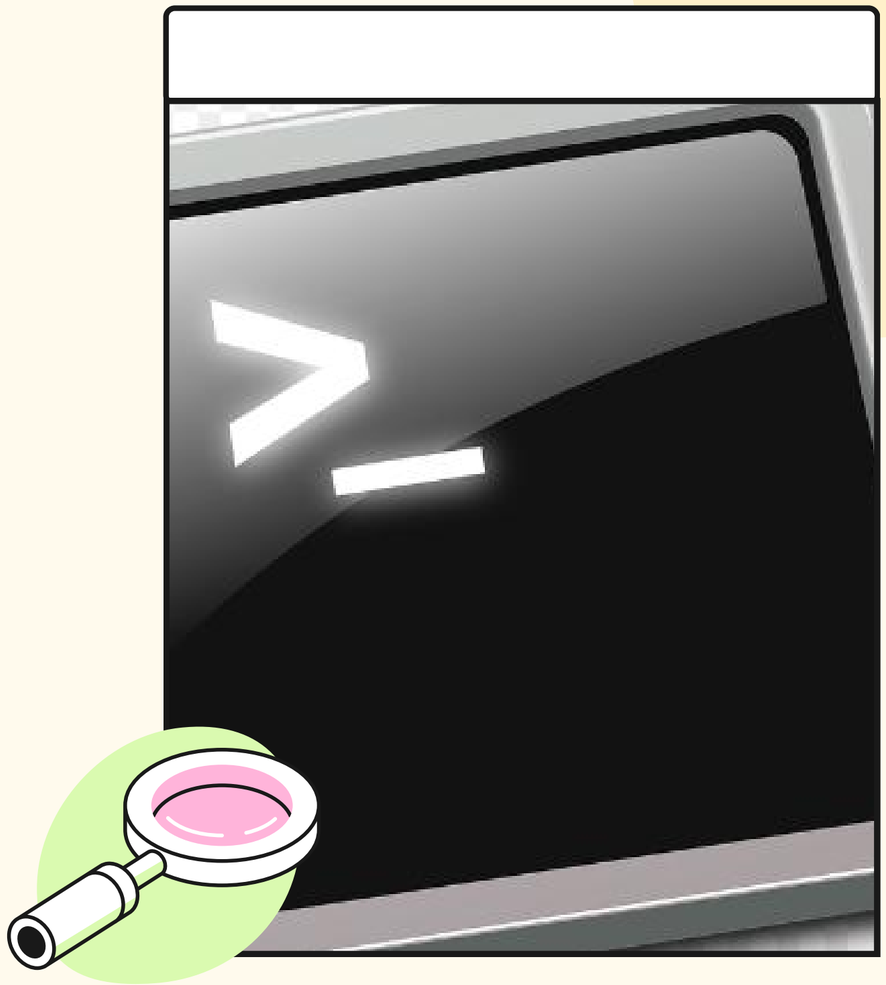
WHAT IT CAN DO?

- Read and print lines
- Do analysis and make charts

PSEUDO CODE

Objective : Write a script that reads a file in a specific location and prints 1st & last line

1. Find command for reading the file
2. Find the command to print the lines
3. Test comman on the CLI
4. Create a script, which can read your "printing_variable" script



1: FINDING COMMAND

Objective : Read a file into Python variable.

file object

An object exposing a file-oriented API (with methods such as `read()` or `write()`) to an underlying resource. Depending on the way it was created, a file object can mediate access to a real on-disk file or to another type of storage or communication device (for example standard input/output, in-memory buffers, sockets, pipes, etc.). File objects are also called *file-like objects* or *streams*.

There are actually three categories of file objects: raw binary files, buffered binary files and text files. Their interfaces are defined in the `io` module. The canonical way to create a file object is by using the `open()` function.

- https://docs.python.org/3/tutorial/inputoutput.html
- There are 3 different file objects,
  - raw binary
  - binary
  - text

Built-in Functions

A	E	L	R
abs()	enumerate()	len()	range()
aiter()	eval()	list()	repr()
all()	exec()	locals()	reversed()
any()			round()
anext()	F	M	
ascii()	filter()	map()	S
	float()	max()	set()
B	format()	memoryview()	setattr()
bin()	frozenset()	min()	slice()
bool()			sorted()
breakpoint()	G	N	staticmethod()
bytearray()	getattr()	next()	str()
bytes()	globals()		sum()
		O	super()
C	H	object()	
callable()	hasattr()	oct()	T
chr()	hash()	open()	tuple()
classmethod()	help()	ord()	type()
compile()	hex()		
complex()		P	V
	I	pow()	vars()
D	id()	print()	
delattr()	input()	property()	Z
dict()	int()		zip()
dir()	isinstance()		
divmod()	issubclass()		_
	iter()		__import__()

COMMAND USAGE

2 ways to use Open() function

Bad !!!

```
fileObj1 = open('file1.txt',mode='r')
```

Good ???

```
with open('dat2.py', '+w) as fileObj2:
```

COMMAND USAGE

How to print the content?
Find how to read, first

Surprised !!!

```
print(fileObj1)
```

Objects can be printed too.

Good ???

```
data = fileObj2.read()
print(data)
```

METHODS COMMAND

Objective : Use the methods to access the file object

- fileObj.read()
- fileObj.readline()
- fileObj.readlines()
- fileObj.write()

How to read all the lines?

```
for line in f:
    print(line, end = '\n')
```

It is good practice to use the `with` keyword when dealing with file objects. The advantage is that the file is properly closed after its suite finishes, even if an exception is raised at some point. Using `with` is also much shorter than writing equivalent `try-finally` blocks:

```
>>> with open('workfile', encoding="utf-8") as f:
...     read_data = f.read()

>>> # We can check that the file has been automatically closed.
>>> f.closed
True
```

Lets go to the Terminal

print_file.py

```
#!/usr/bin/env python
#As shown in the video use the
#commands and create your script and
#execute
```

COMMAND LINE

```
python print_file.py
```

WE HAVE FILES,
LETS DO DATA STRUCTURES!!!!

Any Questions...