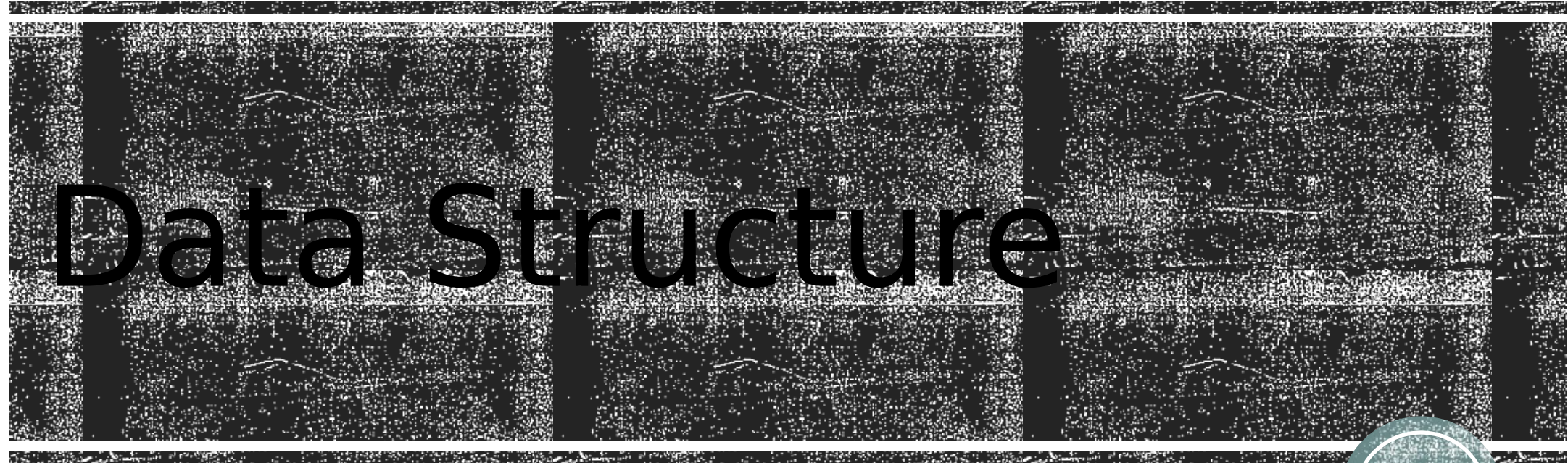


Review: Dictionary

A dictionary is a mapping between KEYS and set of Values.

key-value pair



Review

```
>>> eng2sp['one'] = 'uno'
```

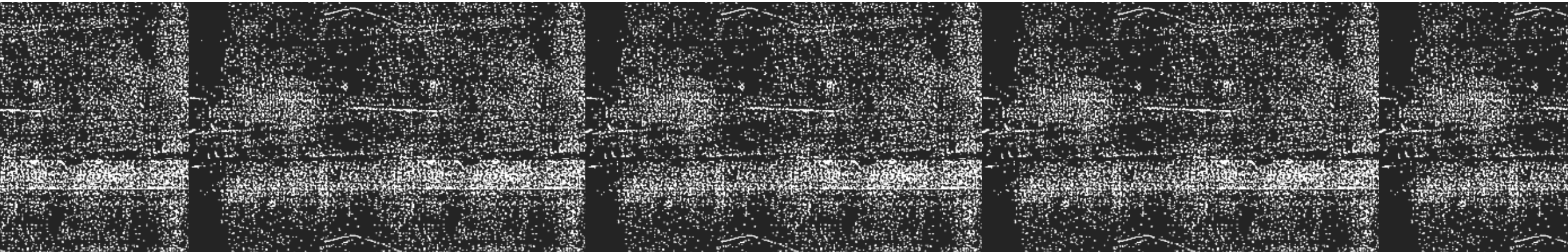
```
>>> print(eng2sp)
```

```
{'one': 'uno'}
```





Tuples



Tuples

- a sequence of values
- are indexed by integers

```
>>> t = 'a', 'b', 'c'
```



Tuples

- Tuples are Immutable

```
>>> t = ('a', 'b', 'c')
```



Tuples

```
>>> t1 = ()
```

```
>>> t2 = (1,)
```



Tuple Assignment

a, *b* = 2, 3



Return values

```
def divmod(a,b):  
    return a//b, a%b
```



Dictionaries & Tuples

```
>>> d = {'a': 12, 'b': 18}
```

```
>>> t = d.items()
```

```
[('b', 18), ('a', 12)]
```



Looping

```
for key, value in d.items():  
    print(key, '==>', value)
```



Files

Lab07: File and Exceptions



Input
and
Output
Devices



Software

Central

Processin
g
Unit

What
Next?



Secondary
Memory



if $x < 3$:
print

Main
Memory



Persistence?

- Programs we wrote so far: **Transient**
- Others: **Persistent**
 - Web servers
 - Operating Systems



File

Every computer system uses **FILES** to save things from one computation to the other



File Processing

Simplest way for programs to maintain their data is by **reading** and **writing** text files



File processing

Each operating system comes with its own file system for creating and accessing files but Python achieves OS-independence by accessing files through something called **FILE HANDLE**.



Opening a File

Before we can **read/write** the contents of the file we must tell Python which file we are going to work with and what we will be doing with the file



Opening a File

- This is done with the `open()` function
- `open()` returns a “file handle” - a variable used to perform operations on the file



open()

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

- filename is a string
- mode is optional and should be 'r' if we are planning reading the file and 'w' if we are going to write to the file
- returns a handle use to manipulate the file



open modes

- r
- w
- a
- rb
- wb



Text file processing

- A text file can be thought of as a sequence of lines
- A text file has **newlines** at the end of each line



File handle as a sequence

- A file handle open for read can be treated as a **sequence** of strings where each line in the file is a string in the sequence
- We can use the for statement to iterate through a sequence
- Remember - a sequence is an ordered set



File handle as a sequence

```
xfile = open('mbox.txt', 'r')  
for line in xfile:  
    print(line)  
xfile.close()
```



`open()`

`mode read('r')`

`fh.read()`

`fh.readline()`

`fh.readlines()`

`mode write('w')`

`fh.write(s)`

`fh.writelines(l)`



close()



IOError: [Errno 2]

```
>>> fhand = open('stuff.txt')
```

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

IOError: [Errno 2]

No such file or directory: 'stuff.txt'



PermissionError: [Errno 13]

```
>>> fout = open('/etc/passwd', 'w')
```

PermissionError: [Errno 13]

Permission denied: '/etc/passwd'



Exception

- Exception is 'something that does not conform to the norm'
- **The error message:** STACK TRACEBACK
or TRACEBACK



Exceptions

- Exceptions are RUN TIME ERRORS
- Unhandled exception: an exception that causes a program to terminate... raise



Exception Handling

- Dealing with the exception

Code that may raise an exception

try:

<body>

except <exceptionType>:

<handle>



Sample code

```
success_failure_ratio = num_success/num_failure  
print('success/failure = ', success_failure_ratio)
```

What problem do you anticipate?



Division by zero

try:

```
    success_failure_ratio = num_success/num_failure
```

```
    print('success/failure = ', success_failure_ratio)
```

```
except ZeroDivisionError:
```

```
    print('Division by zero')
```



Back to opening file

try:

 open(file_name, 'r')

except IOError:

 print('File', file_name, 'does not exist')



The try ... except Clause

try:

<body>

except <exceptionType1>:

<handle1>

except <exceptionType2> as e:

<handle2>

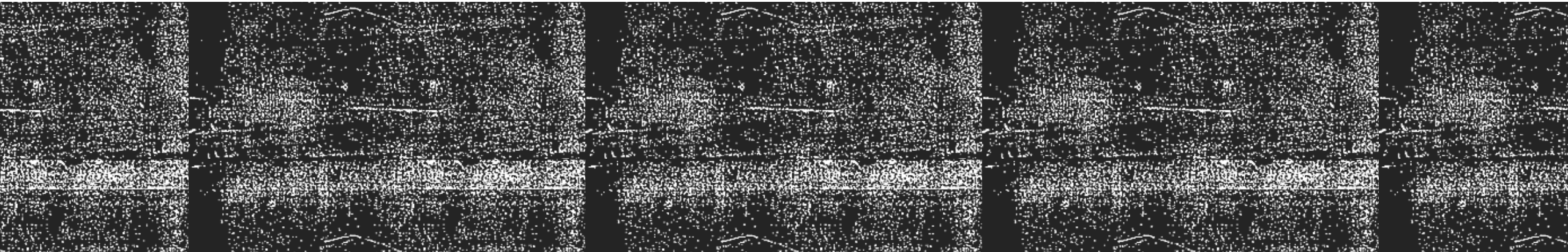
finally:

<finalize_process>





Best practices



Defensive programming

```
x,y = eval(input('Enter integers x,y for the ratio x/y: '))
```

```
try:
```

```
    print('x,y = ', x/y)
```

```
except ZeroDivisionError:
```

```
    print('File', file_name, 'does not exist')
```

```
except:
```

```
    raise ValueError('Bad Arguments')
```



Defensive programming

```
x,y = eval(input('Enter x,y for the ratio x/y: '))
assert type(x)==int and type(y) == int
try:
    print('x,y = ', x/y)
except ZeroDivisionError:
    print('File', file_name, 'does not exist')
```



Unit testing

```
if __name__ == '__main__':  
    test()
```

