Input from user

```
something = input("Enter text: ")
```

io_input.py

```python
def reverse(text):
    return text[::-1]


def is_palindrome(text):
    return text == reverse(text)


something = input("Enter text: ")
if is_palindrome(something):
    print("Yes, it is a palindrome")
else:
    print("No, it is not a palindrome")
```

```
$ python3 io_input.py
Enter text: sir
No, it is not a palindrome

$ python3 io_input.py
Enter text: madam
Yes, it is a palindrome

$ python3 io_input.py
Enter text: racecar
Yes, it is a palindrome
```

Files

io_using_file.py

```python
poem = '''\
Programming is fun
When the work is done
if you wanna make your work also fun:
    use Python!
'''

# Open for 'w'riting
f = open('poem.txt', 'w')
# Write text to file
f.write(poem)
# Close the file
f.close()

# If no mode is specified,
# 'r'ead mode is assumed by default
f = open('poem.txt')
while True:
    line = f.readline()
    # Zero length indicates EOF
    if len(line) == 0:
        break
    # The `line` already has a newline
    # at the end of each line
    # since it is reading from a file.
    print(line, end='')
# close the file
f.close()
```

```
f = open(‘test.txt’,  ‘w’)
f.write(‘text message1’)
...
f.close
```

```
f = open(‘text.txt’, ’r’)
xtxt = f.readline()
...
f.close
```

Output:

```
$ python3 io_using_file.py
Programming is fun
When the work is done
if you wanna make your work also fun:
    use Python!
```

```
import pickle
```

io_pickle.py

```python
import pickle

# The name of the file where we will store the object
shoplistfile = 'shoplist.data'
# The list of things to buy
shoplist = ['apple', 'mango', 'carrot']

# Write to the file
f = open(shoplistfile, 'wb')
# Dump the object to a file
pickle.dump(shoplist, f)
f.close()

# Destroy the shoplist variable
del shoplist

# Read back from the storage
f = open(shoplistfile, 'rb')
# Load the object from the file
storedlist = pickle.load(f)
print(storedlist)
```

```
pickle.dump(shoplist, f)
```

```
storedlist = pickle.load(f)
```

Output:

```
$ python io_pickle.py
['apple', 'mango', 'carrot']
```

Unicode

```
>>> u"hello world"
'hello world'
>>> type(u"hello world")
<class 'str'>
```

```
>>> "hello world"
'hello world'
>>> type("hello world")
<class 'str'>
```

```
# encoding=utf-8
import io

f = io.open("abc.txt", "wt", encoding="utf-8")
f.write(u"Imagine non-English language here")
f.close()

text = io.open("abc.txt", encoding="utf-8").read()
print(text)
```

NOTE: If you are using Python 2, and we want to be able to read and write other non-English languages, we need to use the `unicode` type, and it all starts with the characte `u`, e.g. `u"hello world"`

Errors

```
>>> Print("Hello World")
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'Print' is not defined
>>> print("Hello World")
Hello World
```

Exceptions

```
>>> s = input('Enter something --> ')
Enter something --> Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
EOFError
```

exceptions_handle.py

```python
try:
    text = input('Enter something --> ')
except EOFError:
    print('Why did you do an EOF on me?')
except KeyboardInterrupt:
    print('You cancelled the operation.')
else:
    print('You entered {}'.format(text))
```

```
# Press ctrl + d
$ python exceptions_handle.py
Enter something --> Why did you do an EOF on me?
```

```
# Press ctrl + c
$ python exceptions_handle.py
Enter something --> ^CYou cancelled the operation.
```

```
$ python exceptions_handle.py
Enter something --> No exceptions
You entered No exceptions
```

exceptions_raise.py

```python
class ShortInputException(Exception):
    '''A user-defined exception class.'''
    def __init__(self, length, atleast):
        Exception.__init__(self)
        self.length = length
        self.atleast = atleast

try:
    text = input('Enter something --> ')
    if len(text) < 3:
        raise ShortInputException(len(text), 3)
    # Other work can continue as usual here
except EOFError:
    print('Why did you do an EOF on me?')
except ShortInputException as ex:
    print(('ShortInputException: The input was ' +
           '{0} long, expected at least {1}')
          .format(ex.length, ex.atleast))
else:
    print('No exception was raised.')
```

```
$ python exceptions_raise.py
Enter something --> a
ShortInputException: The input was 1 long, expected at least 3
```

```
$ python exceptions_raise.py
Enter something --> abc
No exception was raised.
```

exceptions_finally.py

```python
import sys
import time

f = None
try:
    f = open("poem.txt")
    # Our usual file-reading idiom
    while True:
        line = f.readline()
        if len(line) == 0:
            break
        print(line, end='')
        sys.stdout.flush()
        print("Press ctrl+c now")
        # To make sure it runs for a while
        time.sleep(2)
except IOError:
    print("Could not find file poem.txt")
except KeyboardInterrupt:
    print("!! You cancelled the reading from the file.")
finally:
    if f:
        f.close()
    print("(Cleaning up: Closed the file)")
```

```
$ python exceptions_finally.py
Programming is fun
Press ctrl+c now
^C!! You cancelled the reading from the file.
(Cleaning up: Closed the file)
```

with statement

`thefile.__enter__`   `thefile.__exit__`

`exceptions_using_with.py`

```python
with open("poem.txt") as f:
    for line in f:
        print(line, end='')
```

How It Works

The output should be same as the previous example. The difference here is that we are using the `open` function with the `with` statement - we leave the closing of the file to be done automatically by `with open`.

What happens behind the scenes is that there is a protocol used by the `with` statement. It fetches the object returned by the `open` statement, let's call it "thefile" in this case.

It *always* calls the `thefile.__enter__` function before starting the block of code under it and *always* calls `thefile.__exit__` after finishing the block of code.

So the code that we would have written in a `finally` block should be taken care of automatically by the `__exit__` method. This is what helps us to avoid having to use explicit `try..finally` statements repeatedly.

More discussion on this topic is beyond scope of this book, so please refer PEP 343 for a comprehensive explanation.