CONTENT

Console and File Handling

JSON and XML Handling

Error/Exception Handling

Problem Solving : Read/Write Object Detection

Console and File Handling

- *Console (also called Shell) is basically a command line interpreter that takes input from the user i.e one command at a time and interprets it. If it is error free then it runs the command and gives required output otherwise shows the error message.*
- *User enters the values in the Console and that value is then used in the program as it was required.*
- *To take input from the user we make use of a built-in function input().*

Console and File Handling

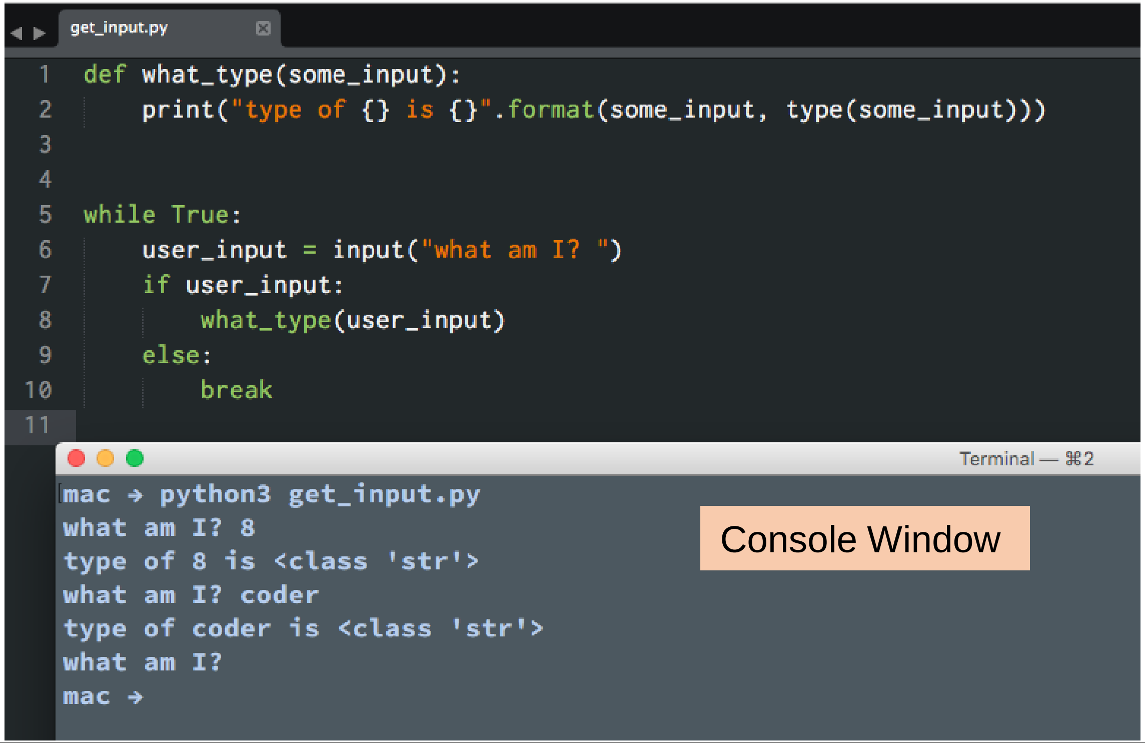
FPT Education
FPT UNIVERSITY

- *Example:*

```
num1 = int(input())
num2 = float(input())
string = str(input())
# printing the sum in integer
print(string, num1 + num2)
```

Console and File Handling

Input a value



get_input.py

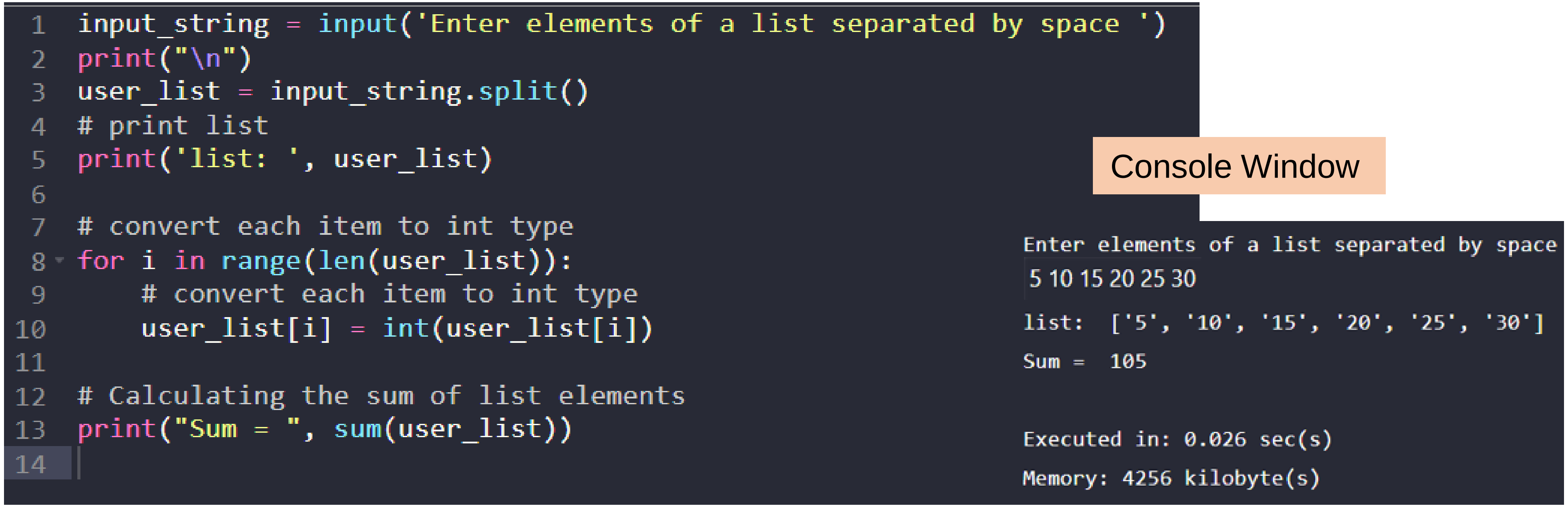
```python
def what_type(some_input):
    print("type of {} is {}".format(some_input, type(some_input)))


while True:
    user_input = input("what am I? ")
    if user_input:
        what_type(user_input)
    else:
        break
```

```
mac → python3 get_input.py
what am I? 8
type of 8 is <class 'str'>
what am I? coder
type of coder is <class 'str'>
what am I?
mac →
```

Console Window

Console and File Handling

Input a List of values



```python
input_string = input('Enter elements of a list separated by space ')
print("\n")
user_list = input_string.split()
# print list
print('list: ', user_list)

# convert each item to int type
for i in range(len(user_list)):
    # convert each item to int type
    user_list[i] = int(user_list[i])

# Calculating the sum of list elements
print("Sum = ", sum(user_list))
```

Console Window

```
Enter elements of a list separated by space
5 10 15 20 25 30

list:  ['5', '10', '15', '20', '25', '30']
Sum =  105


Executed in: 0.026 sec(s)
Memory: 4256 kilobyte(s)
```

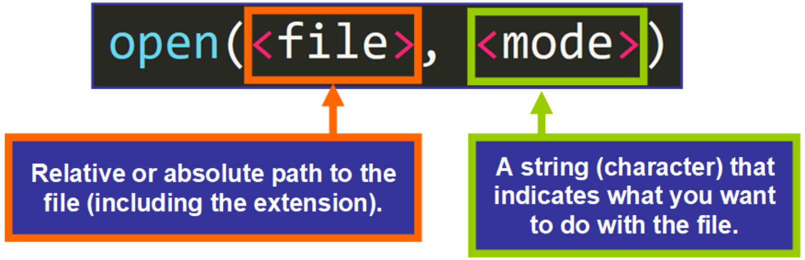
Console and File Handling

- *File is a named location on disk to store related information. It is used to permanently store data in a non-volatile memory (e.g. hard disk).*
- *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 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*
- *Example f = open("test.txt") # open file in current directory*

  f = open("C:/Python33/README.txt") # specifying full path

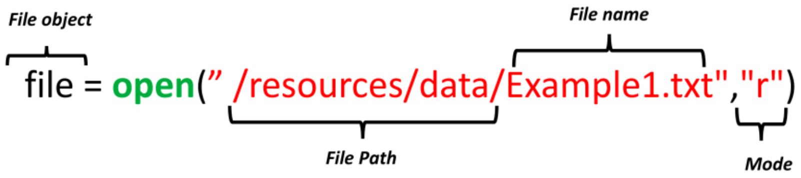
Console and File Handling

Open File Syntax



```
open(<file>, <mode>)
```

Relative or absolute path to the file (including the extension).
A string (character) that indicates what you want to do with the file.



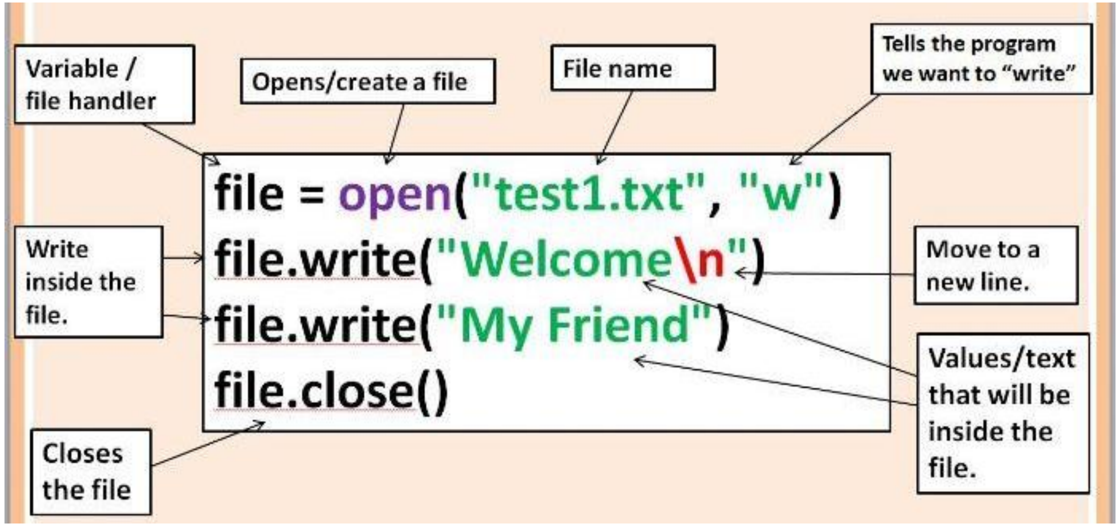
```
file = open(" /resources/data/Example1.txt","r")
```

Open File Mode

Character	Meaning
`'r'`	open for reading (default)
`'w'`	open for writing, truncating the file first
`'x'`	open for exclusive creation, failing if the file already exists
`'a'`	open for writing, appending to the end of the file if it exists
`'b'`	binary mode
`'t'`	text mode (default)
`'+'`	open a disk file for updating (reading and writing)

Console and File Handling

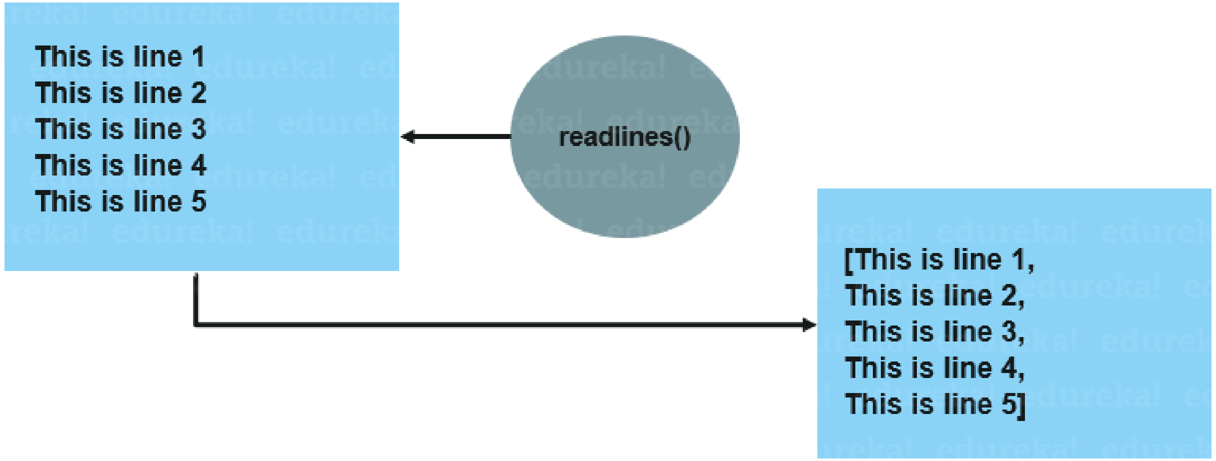
Write to a file



```
file = open("test1.txt", "w")
file.write("Welcome\n")
file.write("My Friend")
file.close()
```

Console and File Handling

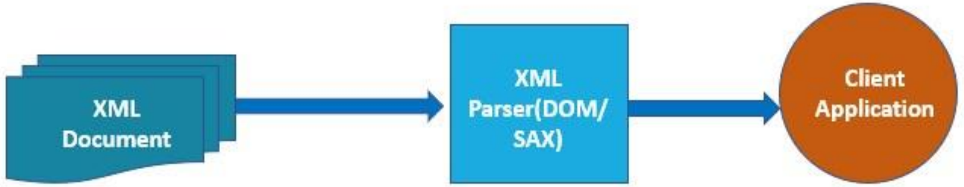
Read all lines from file



```
f = open('alice.txt', 'r')
lines = f.readlines()
print lines
for line in lines:
    print line
f.close()
```

JSON and XML Handling

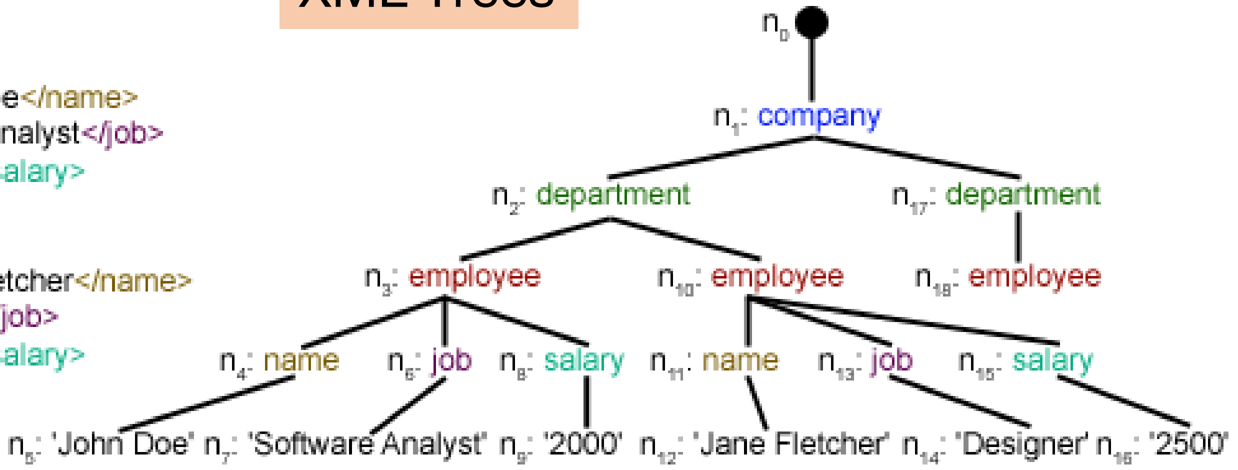
- *XML stands for eXtensible Markup Language. It was designed to store and transport small to medium amounts of data and is widely used for sharing structured information.*
- *Python enables you to parse and modify XML document. In order to parse XML document you need to have the entire XML document in memory. In this tutorial, we will see how we can use XML minidom class in Python to load and parse XML file.*
- *Example*



JSON and XML Handling

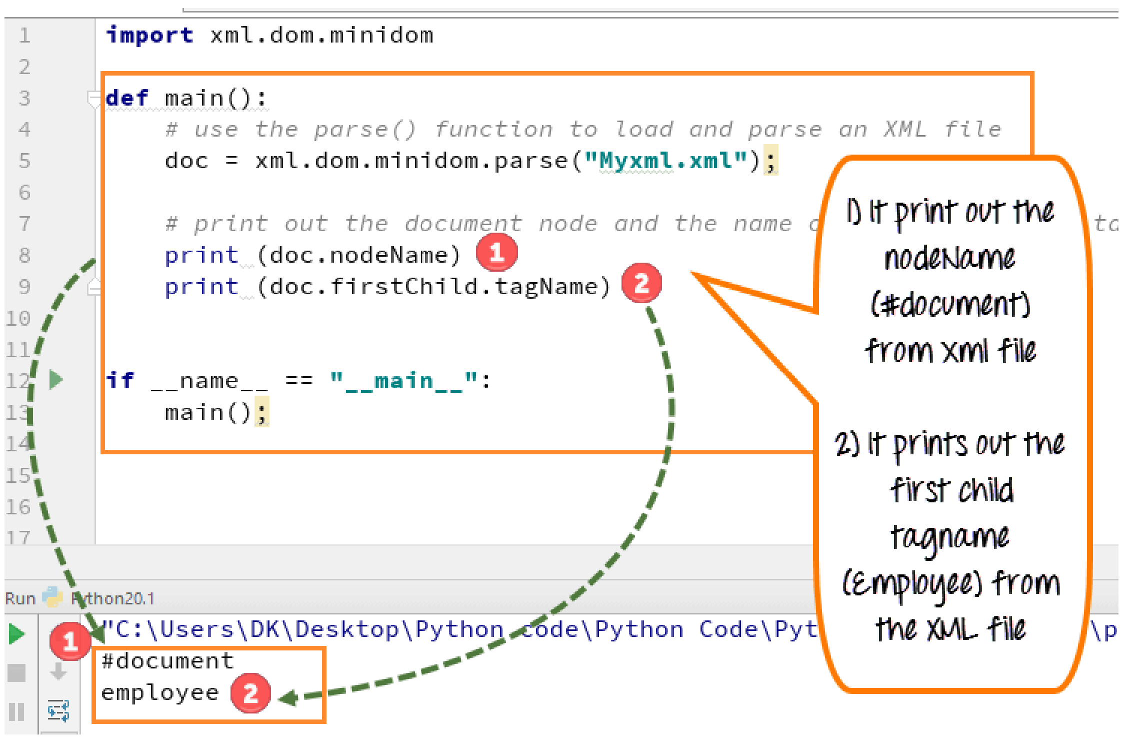
XML Trees

```
<company>
 <department>
  <employee>
   <name>John Doe</name>
   <job>Software Analyst</job>
   <salary>2000</salary>
  </employee>
  <employee>
   <name>Jane Fletcher</name>
   <job>Designer</job>
   <salary>2500</salary>
  </employee>
 </department>
 <department>
  <employee>
  </employee>
 </department>
</company>
```



JSON and XML Handling

Read first node



```
import xml.dom.minidom

def main():
    # use the parse() function to load and parse an XML file
    doc = xml.dom.minidom.parse("Myxml.xml");

    # print out the document node and the name o                          ta
    print (doc.nodeName)   1
    print (doc.firstChild.tagName)   2


if __name__ == "__main__":
    main();
```

Run Python20.1

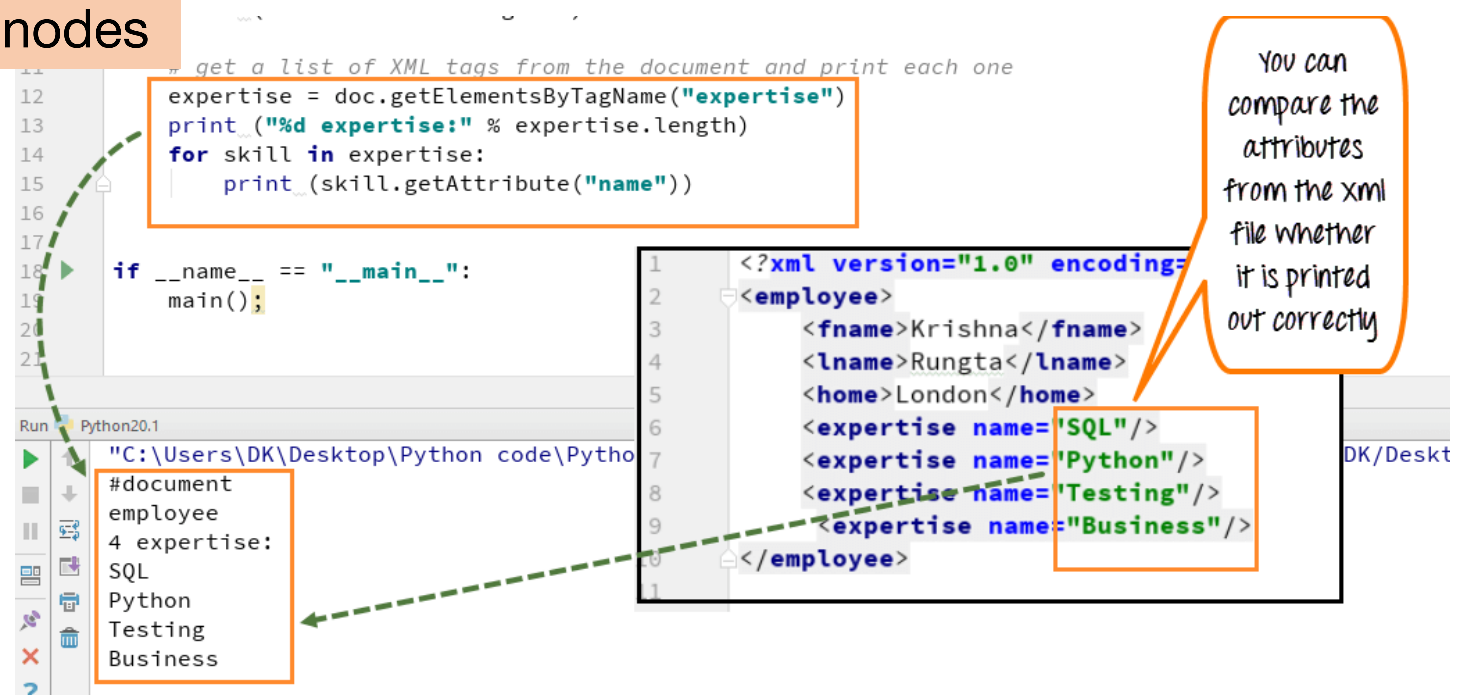
```
"C:\Users\DK\Desktop\Python code\Python Code\Pyt                              \p
#document
employee
```

1) It print out the nodeName (#document) from xml file

2) It prints out the first child tagname (Employee) from the XML file

JSON and XML Handling

Read child nodes



```
# get a list of XML tags from the document and print each one
expertise = doc.getElementsByTagName("expertise")
print ("%d expertise:" % expertise.length)
for skill in expertise:
    print (skill.getAttribute("name"))


if __name__ == "__main__":
    main();
```

Run Python20.1

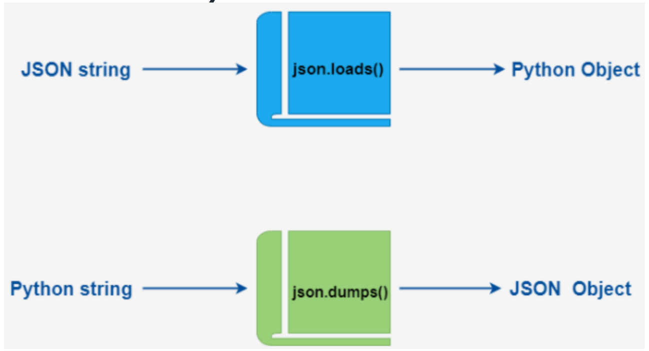
```
"C:\Users\DK\Desktop\Python code\Pytho                    DK/Deskt
#document
employee
4 expertise:
SQL
Python
Testing
Business
```

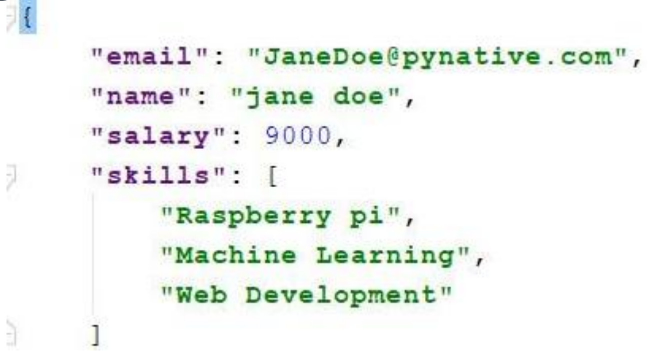
```
<?xml version="1.0" encoding=
<employee>
    <fname>Krishna</fname>
    <lname>Rungta</lname>
    <home>London</home>
    <expertise name="SQL"/>
    <expertise name="Python"/>
    <expertise name="Testing"/>
    <expertise name="Business"/>
</employee>
```

You can compare the attributes from the xml file whether it is printed out correctly

JSON and XML Handling

- *JSON is commonly used for the exchange of data on the web.*
- *More specifically, JSON is the preferred text format when sending information from a web server to a browser or vice versa. This is advantageous simply because of its efficiency. JSON can be directly converted into JavaScript objects and thus interpreted, and JavaScript objects can be directly converted into JSON text.*
- *Example*

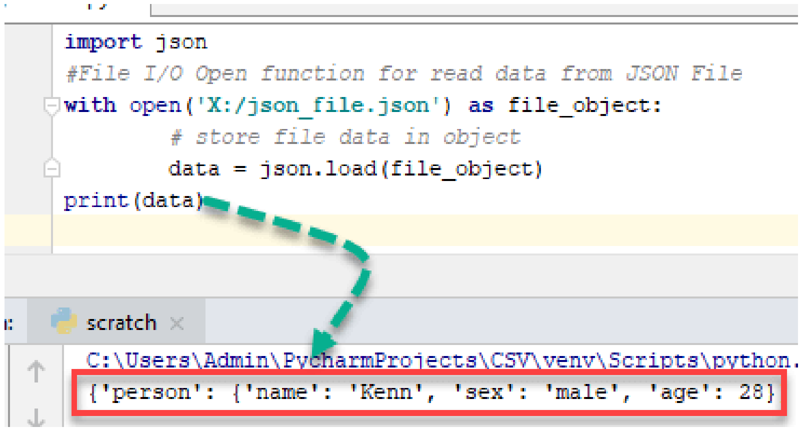




```
{
    "email": "JaneDoe@pynative.com",
    "name": "jane doe",
    "salary": 9000,
    "skills": [
        "Raspberry pi",
        "Machine Learning",
        "Web Development"
    ]
```

JSON and XML Handling

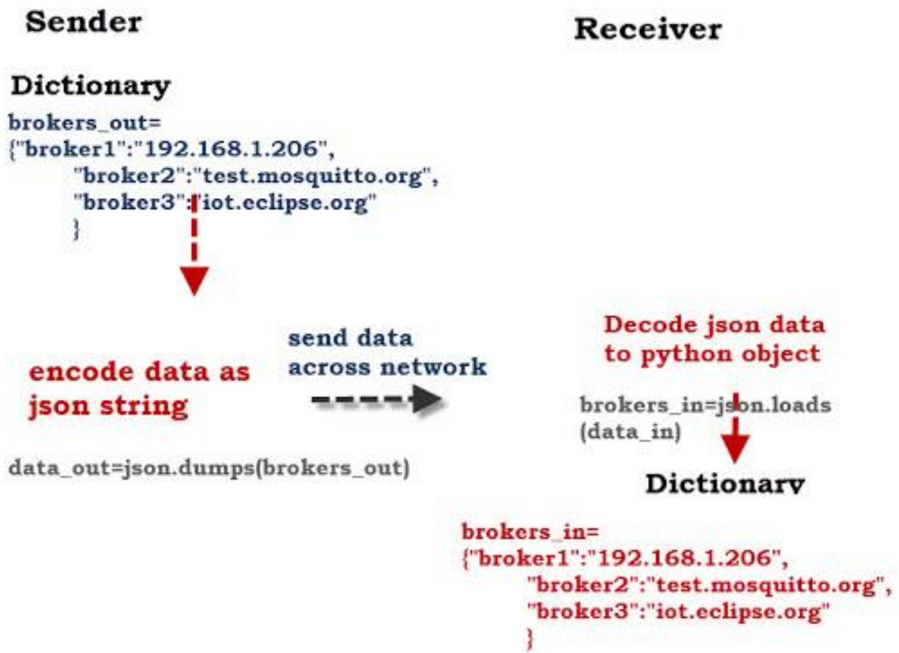
Open Json Files



```
import json
#File I/O Open function for read data from JSON File
with open('X:/json_file.json') as file_object:
        # store file data in object
        data = json.load(file_object)
print(data)
```

```
C:\Users\Admin\PycharmProjects\CSV\venv\Scripts\python.
{'person': {'name': 'Kenn', 'sex': 'male', 'age': 28}
```

Send Json through network



Sender
Receiver
Dictionary
brokers_out=
{"broker1":"192.168.1.206",
"broker2":"test.mosquitto.org",
"broker3":"iot.eclipse.org"
}
encode data as json string
data_out=json.dumps(brokers_out)
send data across network
Decode json data to python object
brokers_in=json.loads (data_in)
Dictionary
brokers_in=
{"broker1":"192.168.1.206",
"broker2":"test.mosquitto.org",
"broker3":"iot.eclipse.org"
}

JSON and XML Handling

FPT Education
FPT UNIVERSITY

Convert JSON to Python Object (Dictionary/Map)

```
import json

json_data = '{"name": "TeraTom", "city": "Seattle"}'
python_obj = json.loads(json_data)
print (python_obj["name"])
print (python_obj["city"])
```

```
TeraTom
Seattle
```

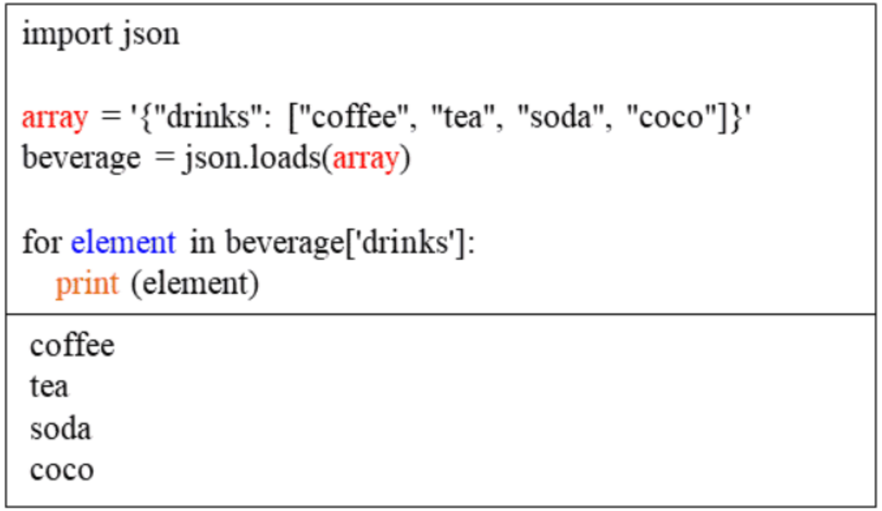
Impressive Printing of JSON

```
import json

json_data = '{"name": "TeraTom", "city": "Seattle"}'
python_obj = json.loads(json_data)
print (json.dumps(python_obj, sort_keys=True, indent=4))
```

```
{
    "city": "Seattle",
    "name": "TeraTom"
}
```

JSON and XML Handling

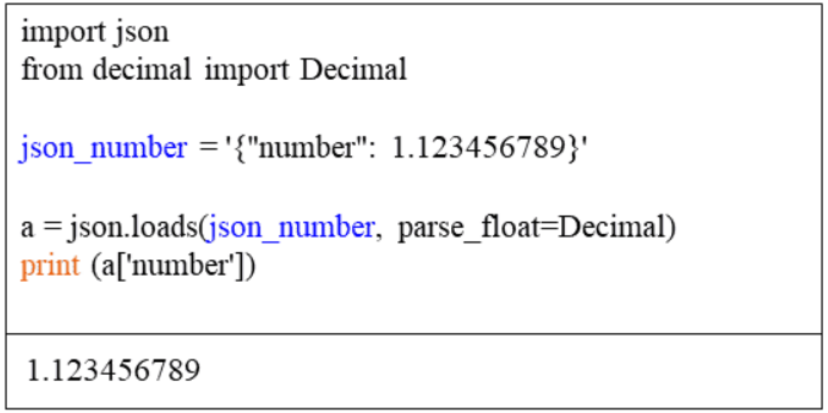
Mapping JSON Data to a List



```
import json

array = '{"drinks": ["coffee", "tea", "soda", "coco"]}'
beverage = json.loads(array)

for element in beverage['drinks']:
    print (element)
```

```
coffee
tea
soda
coco
```

Convert JSON to Python Object (float)

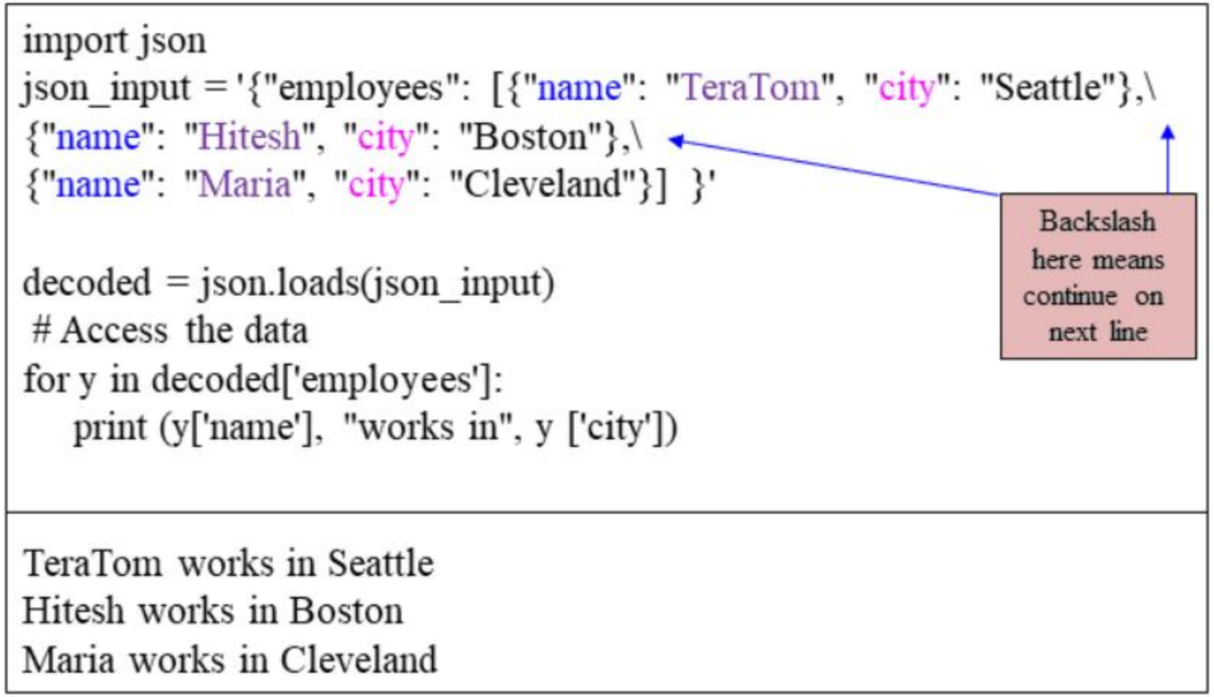


```
import json
from decimal import Decimal

json_number = '{"number": 1.123456789}'

a = json.loads(json_number, parse_float=Decimal)
print (a['number'])
```

```
1.123456789
```

JSON and XML Handling

Using JSON data on multiple objects

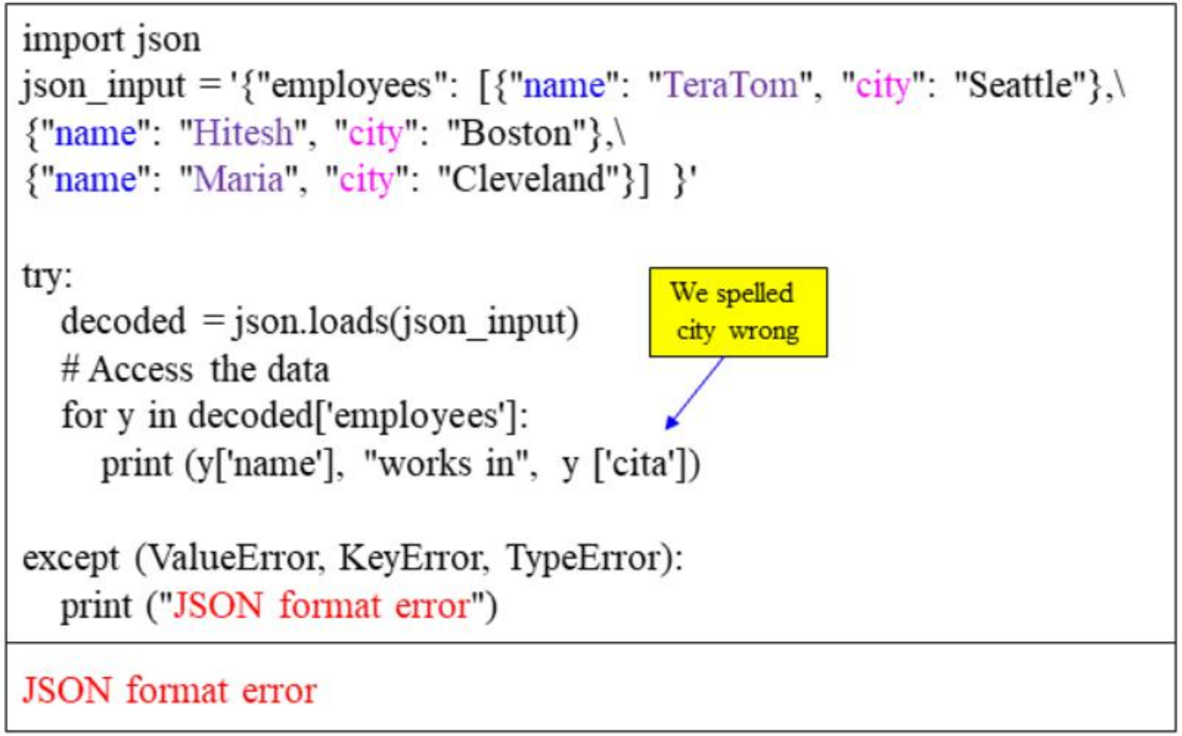


```
import json
json_input = '{"employees":  [{"name":  "TeraTom",  "city":  "Seattle"},\
{"name":  "Hitesh",  "city":  "Boston"},\
{"name":  "Maria",  "city":  "Cleveland"}]  }'

decoded = json.loads(json_input)
 # Access  the data
for y in decoded['employees']:
    print (y['name'],  "works in", y ['city'])
```

```
TeraTom  works in Seattle
Hitesh works in Boston
Maria works in Cleveland
```

JSON and XML Handling

Using JSON data on multiple objects with Error Handling



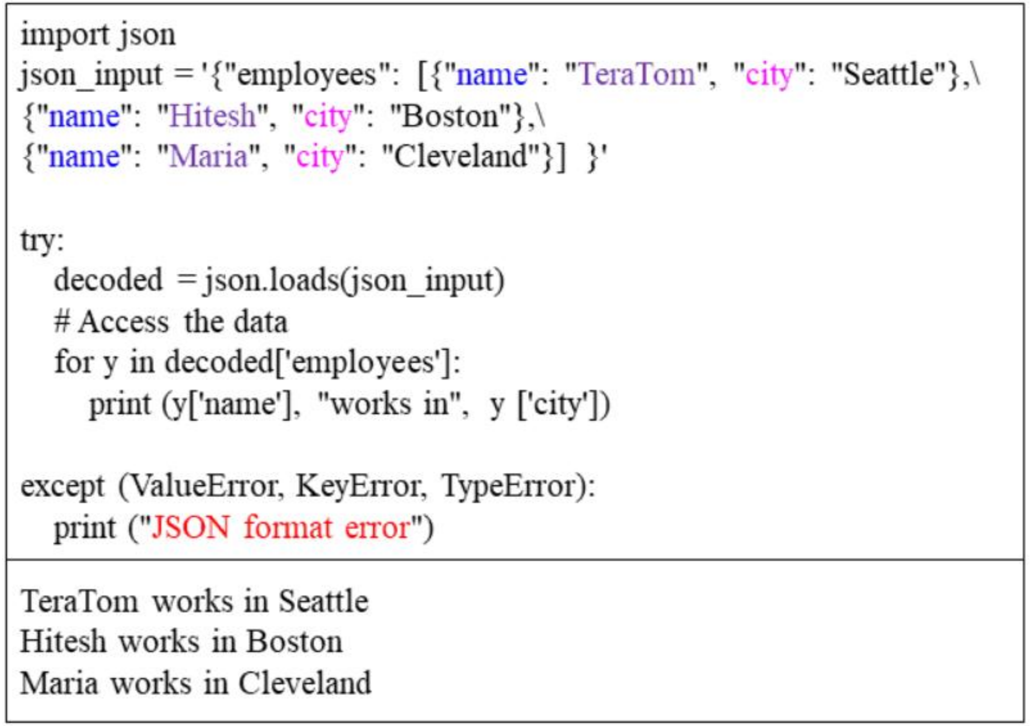
```
import json
json_input = '{"employees": [{"name": "TeraTom", "city": "Seattle"},\
{"name": "Hitesh", "city": "Boston"},\
{"name": "Maria", "city": "Cleveland"}] }'

try:
    decoded = json.loads(json_input)
    # Access the data
    for y in decoded['employees']:
        print (y['name'], "works in", y ['cita'])

except (ValueError, KeyError, TypeError):
    print ("JSON format error")
```

We spelled city wrong

```
JSON format error
```

JSON and XML Handling

JSON data on multiple objects with Error Handling Example

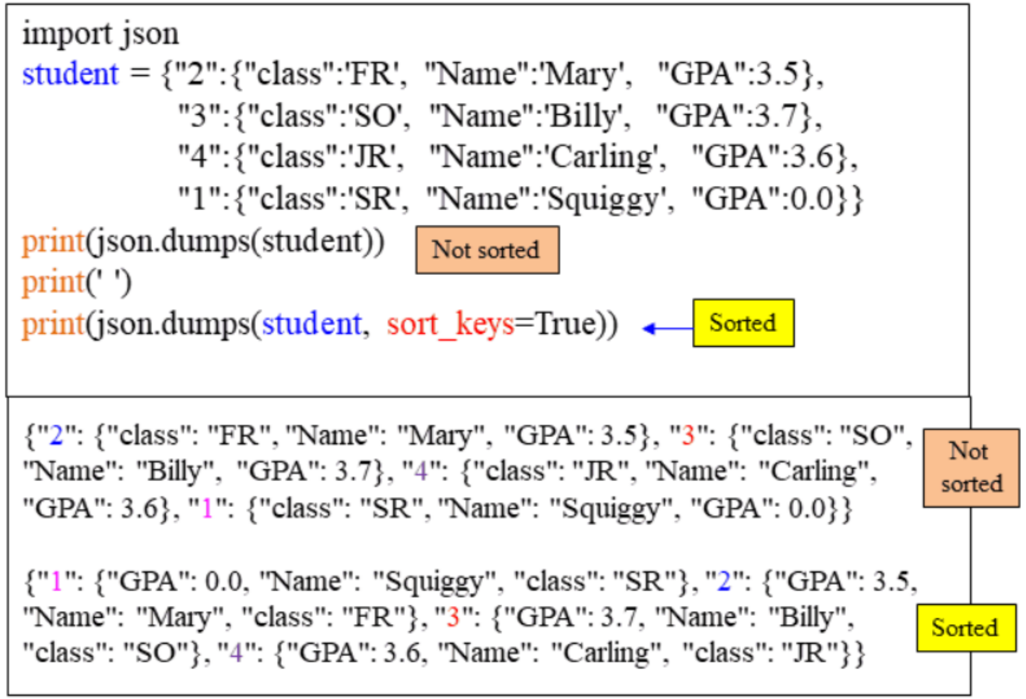


```
import json
json_input = '{"employees": [{"name": "TeraTom", "city": "Seattle"},\
{"name": "Hitesh", "city": "Boston"},\
{"name": "Maria", "city": "Cleveland"}] }'

try:
   decoded = json.loads(json_input)
   # Access the data
   for y in decoded['employees']:
      print (y['name'], "works in", y ['city'])

except (ValueError, KeyError, TypeError):
   print ("JSON format error")
```

```
TeraTom works in Seattle
Hitesh works in Boston
Maria works in Cleveland
```

JSON and XML Handling

Python Dictionaries to JSON Strings with Sorting

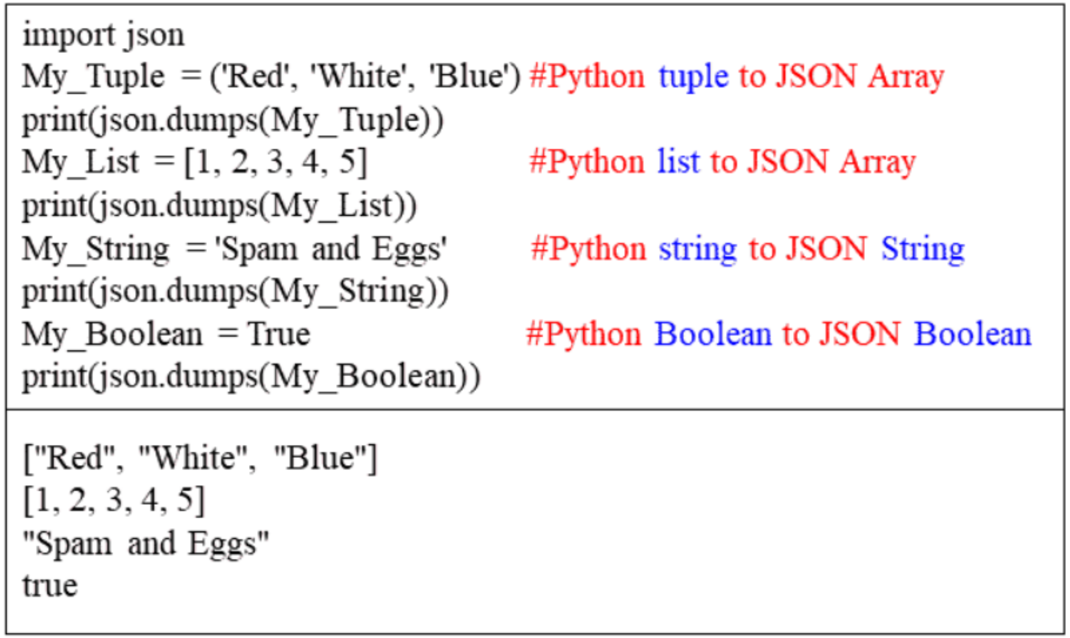


```
import json
student = {"2":{"class":'FR',  "Name":'Mary',   "GPA":3.5},
                "3":{"class":'SO',  "Name":'Billy',   "GPA":3.7},
                "4":{"class":'JR',  "Name":'Carling',   "GPA":3.6},
                "1":{"class":'SR',  "Name":'Squiggy',  "GPA":0.0}}
print(json.dumps(student))
print(' ')
print(json.dumps(student,  sort_keys=True))
```

```
{"2": {"class": "FR", "Name": "Mary", "GPA": 3.5}, "3": {"class": "SO",
"Name": "Billy", "GPA": 3.7}, "4": {"class": "JR", "Name": "Carling",
"GPA": 3.6}, "1": {"class": "SR", "Name": "Squiggy", "GPA": 0.0}}

{"1": {"GPA": 0.0, "Name": "Squiggy", "class": "SR"}, "2": {"GPA": 3.5,
"Name": "Mary", "class": "FR"}, "3": {"GPA": 3.7, "Name": "Billy",
"class": "SO"}, "4": {"GPA": 3.6, "Name": "Carling", "class": "JR"}}
```

JSON and XML Handling

Python Objects to JSON



```
import json
My_Tuple  = ('Red', 'White', 'Blue') #Python tuple to JSON Array
print(json.dumps(My_Tuple))
My_List  = [1, 2, 3, 4, 5]            #Python list to JSON Array
print(json.dumps(My_List))
My_String  = 'Spam and Eggs'          #Python string to JSON String
print(json.dumps(My_String))
My_Boolean  = True                    #Python Boolean to JSON Boolean
print(json.dumps(My_Boolean))
```

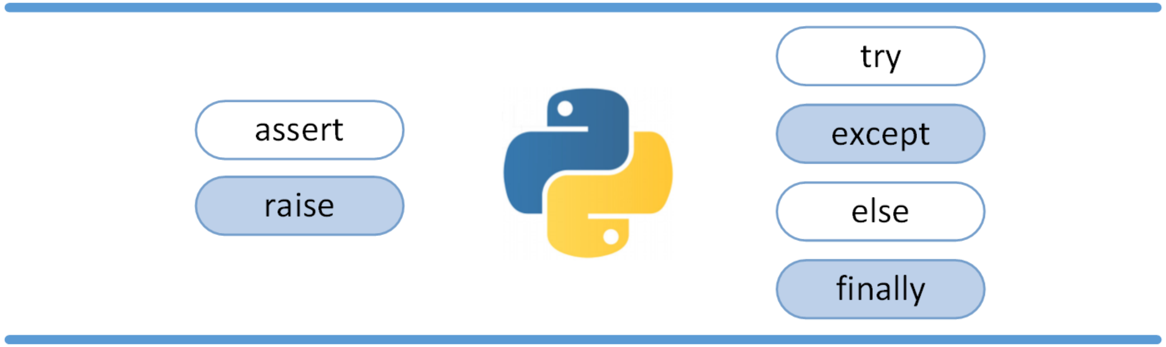
```
["Red", "White", "Blue"]
[1, 2, 3, 4, 5]
"Spam and Eggs"
true
```

Error/Exception Handling

- Errors – the developer's daily bread
- Dealing with programming errors has (at least) two sides.
  - The one appears when you get into trouble because your – apparently correct – code is fed with bad data
  - The other side of dealing with programming errors reveals itself when undesirable code behavior is caused by mistakes you made when you were writing your program.

Error/Exception Handling

- *A Python program terminates as soon as it encounters an error. In Python, an error can be a syntax error or an exception.*
- *Syntax errors occur when the parser detects an incorrect statement*
- *We can use raise to throw an exception if a condition occurs. The statement can be complemented with a custom exception.*



Error/Exception Handling

Use raise to force an exception:

raise
Exception

Python

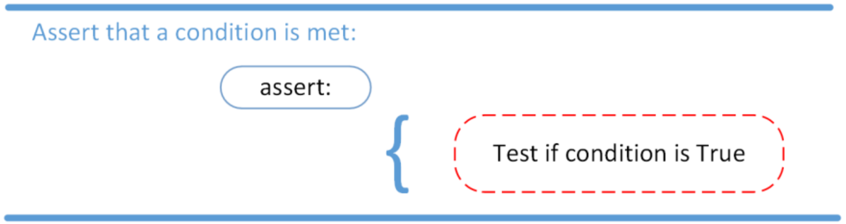
```python
x = 10
if x > 5:
    raise Exception('x should not exceed 5. The value of x was: {}'.format(x))
```

Python Traceback

```
Traceback (most recent call last):
  File "<input>", line 4, in <module>
Exception: x should not exceed 5. The value of x was: 10
```

Error/Exception Handling

- *Instead of waiting for a program to crash midway, you can also start by making an assertion in Python. We assert that a certain condition is met. If this condition turns out to be True, then that is excellent! The program can continue. If the condition turns out to be False, you can have the program throw an AssertionError exception.*

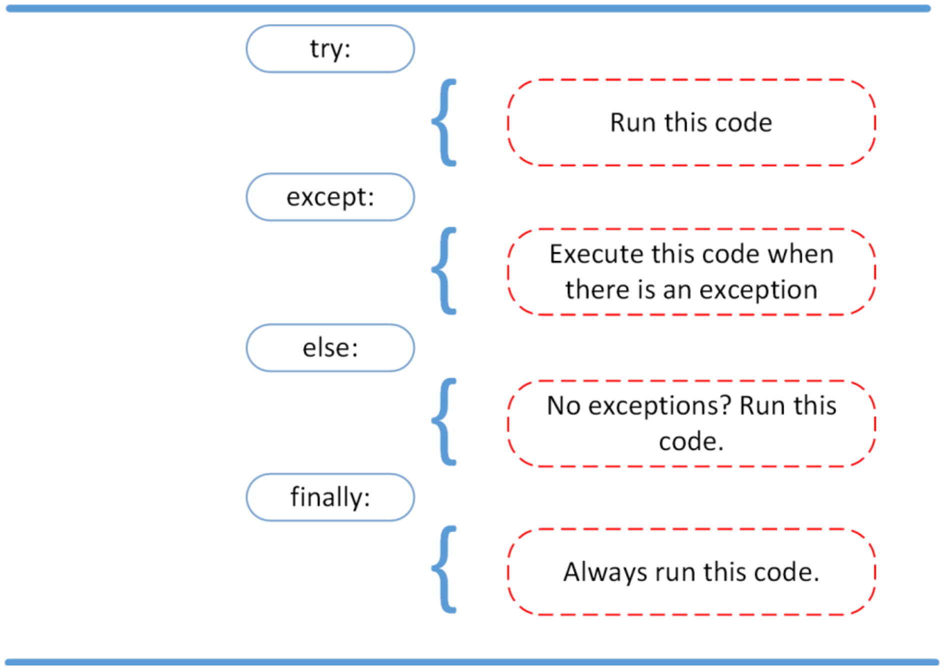


```
Traceback (most recent call last):
  File "<input>", line 2, in <module>
AssertionError: This code runs on Linux only.
```

```
import sys
assert ('linux' in sys.platform), "This code runs on Linux only."
```

Error/Exception Handling

- *The try and except block in Python is used to catch and handle exceptions. Python executes code following the try statement as a "normal" part of the program.*
- *The code that follows the except statement is the program's response to any exceptions in the preceding try clause.*


try:
Run this code
except:
Execute this code when there is an exception
else:
No exceptions? Run this code.
finally:
Always run this code.

Error/Exception Handling

Python

```python
def linux_interaction():
    assert ('linux' in sys.platform), "Function can only run on Linux systems."
    print('Doing something.')
```

Python

```python
try:
    linux_interaction()
except:
    print('Linux function was not executed')
```

Shell

```
Linux function was not executed
```

Error/Exception Handling

Python

```python
try:
    linux_interaction()
except AssertionError as error:
    print(error)
    print('The linux_interaction() function was not executed')
```

Shell

```
Function can only run on Linux systems.
The linux_interaction() function was not executed
```

Error/Exception Handling

Python

```python
try:
    with open('file.log') as file:
        read_data = file.read()
except:
    print('Could not open file.log')
```

Shell

```
Could not open file.log
```

Error/Exception Handling

Python

```python
try:
    with open('file.log') as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
```

Shell

```
[Errno 2] No such file or directory: 'file.log'
```

Error/Exception Handling

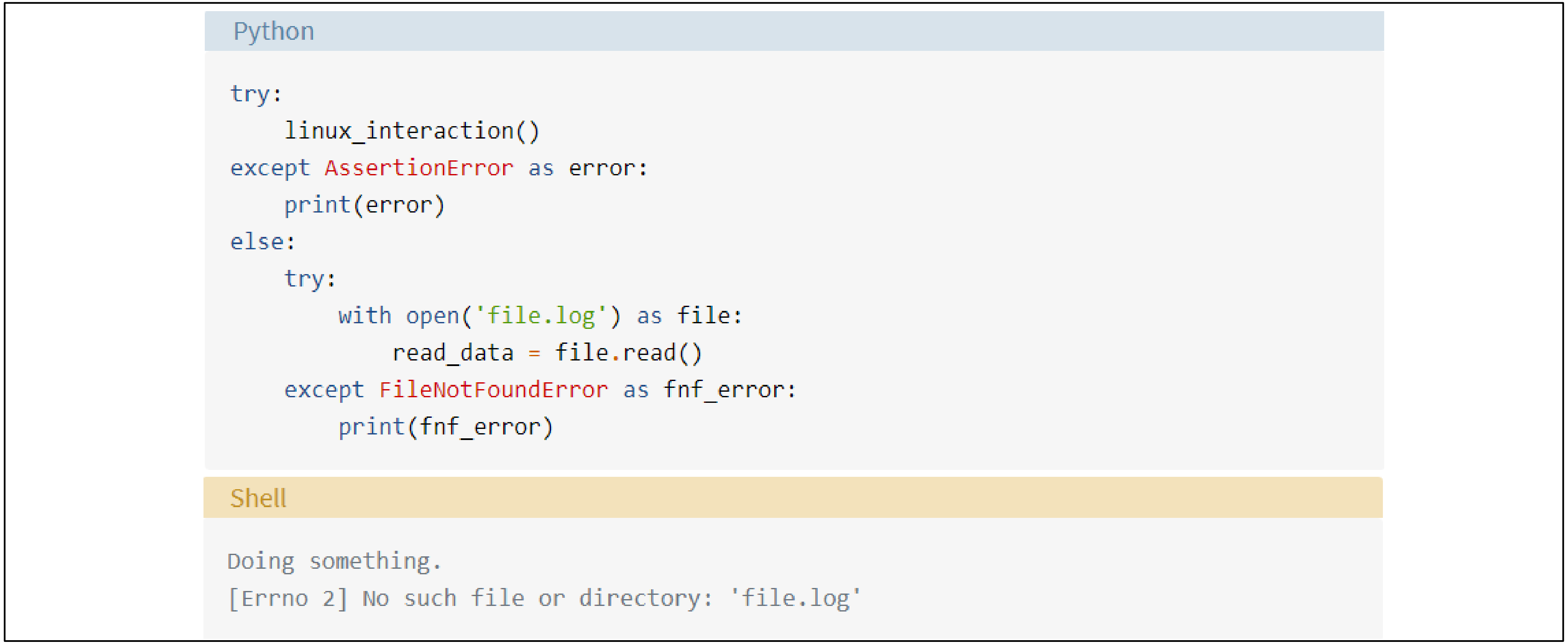
Python

```python
try:
    linux_interaction()
    with open('file.log') as file:
        read_data = file.read()
except FileNotFoundError as fnf_error:
    print(fnf_error)
except AssertionError as error:
    print(error)
    print('Linux linux_interaction() function was not executed')
```

Shell

```
Function can only run on Linux systems.
Linux linux_interaction() function was not executed
```

Error/Exception Handling



Python

```python
try:
    linux_interaction()
except AssertionError as error:
    print(error)
else:
    try:
        with open('file.log') as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
```

Shell

```
Doing something.
[Errno 2] No such file or directory: 'file.log'
```

Error/Exception Handling

Python

```python
try:
    linux_interaction()
except AssertionError as error:
    print(error)
else:
    try:
        with open('file.log') as file:
            read_data = file.read()
    except FileNotFoundError as fnf_error:
        print(fnf_error)
finally:
    print('Cleaning up, irrespective of any exceptions.')
```

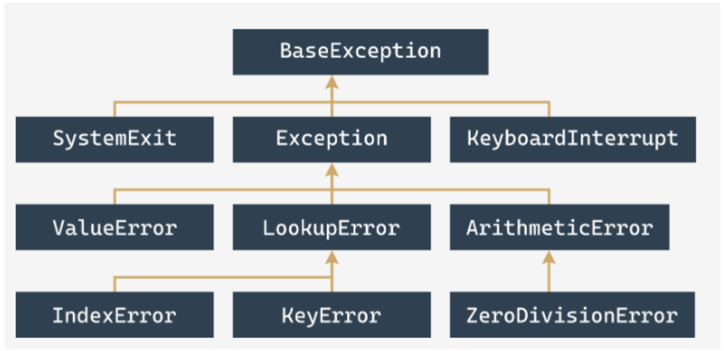
Shell

```
Function can only run on Linux systems.
Cleaning up, irrespective of any exceptions.
```

Error/Exception Handling

- Python 3 defines 63 built-in exceptions, and all of them form a tree-shaped hierarchy, although the tree is a bit weird as its root is located on top.
- The Python statement raise ExceptionName can raise an exception on demand. The same statement, but lacking ExceptionName, can be used inside the try branch only, and raises the same exception which is currently being handled.
- The Python statement assert expression evaluates the expression and raises the AssertError exception when the expression is equal to zero, an empty string, or None. You can use it to protect some critical parts of your code from devastating data.

```
try:
    # Risky code.
except IndexError:
    # Taking care of mistreated lists
except LookupError:
    # Dealing with other erroneous lookups
```



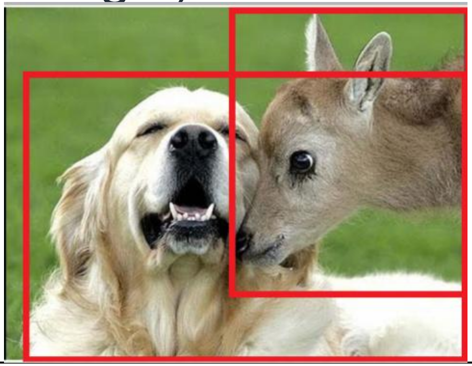
Some exceptions

- ZeroDivisionError: This appears when you try to force Python to perform any operation which provokes division in which the divider is zero, or is indistinguishable from zero they are: /, //, and %.
- ValueError : Expect this exception when you're dealing with values which may be inappropriately used in some context (like int() or float())
- TypeError: This exception shows up when you try to apply a data whose type cannot be accepted in the current context
- AttributeError: This exception arrives – among other occasions – when you try to activate a method which doesn't exist in an item you're dealing with
- SyntaxError: This exception is raised when the control reaches a line of code which violates Python's grammar

Problem Solving

Read/Write Object Detection

- COCO Bounding box: (x-top left, y-top left, width, height)
- Pascal VOC Bounding box :(xmin-top left, ymin-top left,xmax-bottom right, ymax-bottom right)



```
<annotation>
        <folder>Kangaroo</folder>
        <filename>00001.jpg</filename>
        <path>./Kangaroo/stock-12.jpg</path>
        <source>
                <database>Kangaroo</database>
        </source>
        <size>
                <width>450</width>
                <height>319</height>
                <depth>3</depth>
        </size>
        <segmented>0</segmented>
        <object>
                <name>kangaroo</name>
                <pose>Unspecified</pose>
                <truncated>0</truncated>
                <difficult>0</difficult>
                <bndbox>
                        <xmin>233</xmin>
                        <ymin>89</ymin>
                        <xmax>386</xmax>
                        <ymax>262</ymax>
                </bndbox>
        </object>
</annotation>
```