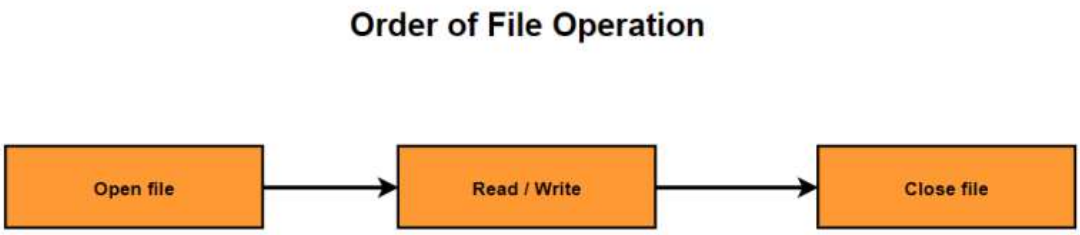
File Management

Python has several built-in modules and functions for creating,reading, updating and deleting files.



Open File

In [115]:
```
fileobj = open('test1.txt') # Open file in read/text mode
```

In [116]:
```
fileobj = open('test1.txt', 'r') # Open file in read mode
```

In [117]:
```
fileobj = open('test1.txt', 'w') # Open file in write mode
```

In [118]:
```
fileobj = open('test1.txt', 'a') # Open file in append mode
```

Close File

In [119]:
```
fileobj.close()
```

Read File

In [120]:
```
fileobj = open('test1.txt')
```

In [122]:
```
fileobj.read() #Read whole file
```

Out[122]:
```
"Python is an interpreted, object-oriented, high-level programming language w
ith dynamic semantics. Its high-level built in data structures, combined with
dynamic typing and dynamic binding, make it very attractive for Rapid Applica
tion Development, as well as for use as a scripting or glue language to conne
ct existing components together.  \n\nPython's simple, easy to learn syntax e
mphasizes readability and therefore reduces the cost of program maintenance.
Python supports modules and packages, which encourages program modularity and
code reuse. \n\nThe Python interpreter and the extensive standard library are
available in source or binary form without charge for all major platforms, an
d can be freely distributed.\n\nOften, programmers fall in love with Python b
ecause of the increased productivity it provides. Since there is no compilati
on step, the edit-test-debug cycle is incredibly fast. \nDebugging Python pro
grams is easy: a bug or bad input will never cause a segmentation fault.\n\nI
nstead, when the interpreter discovers an error, it raises an exception. When
the program doesn't catch the exception, the interpreter prints a stack trac
e. A source level debugger allows inspection of local and global variables, e
valuation of arbitrary expressions, setting breakpoints, stepping through the
code a line at a time, and so on. \n\nThe debugger is written in Python itsel
f, testifying to Python's introspective power. On the other hand, often the q
uickest way to debug a program is to add a few print statements to the sourc
e: the fast edit-test-debug cycle makes this simple approach very effective."
```

In [123]:
```
fileobj.read() #File cursor is already at the end of the file so it won't be a
```

Out[123]:
```
''
```

In [124]:
```
fileobj.seek(0) # Bring file cursor to initial position.
fileobj.read()
```

Out[124]:
```
"Python is an interpreted, object-oriented, high-level programming language w
ith dynamic semantics. Its high-level built in data structures, combined with
dynamic typing and dynamic binding, make it very attractive for Rapid Applica
tion Development, as well as for use as a scripting or glue language to conne
ct existing components together.  \n\nPython's simple, easy to learn syntax e
mphasizes readability and therefore reduces the cost of program maintenance.
Python supports modules and packages, which encourages program modularity and
code reuse. \n\nThe Python interpreter and the extensive standard library are
available in source or binary form without charge for all major platforms, an
d can be freely distributed.\n\nOften, programmers fall in love with Python b
ecause of the increased productivity it provides. Since there is no compilati
on step, the edit-test-debug cycle is incredibly fast. \nDebugging Python pro
grams is easy: a bug or bad input will never cause a segmentation fault.\n\nI
nstead, when the interpreter discovers an error, it raises an exception. When
the program doesn't catch the exception, the interpreter prints a stack trac
e. A source level debugger allows inspection of local and global variables, e
valuation of arbitrary expressions, setting breakpoints, stepping through the
code a line at a time, and so on. \n\nThe debugger is written in Python itsel
f, testifying to Python's introspective power. On the other hand, often the q
uickest way to debug a program is to add a few print statements to the sourc
e: the fast edit-test-debug cycle makes this simple approach very effective."
```

In [125]:
```
fileobj.seek(7) # place file cursor at loc 7
fileobj.read()
```

Out[125]:
```
"is an interpreted, object-oriented, high-level programming language with dyn
amic semantics. Its high-level built in data structures, combined with dynami
c typing and dynamic binding, make it very attractive for Rapid Application D
evelopment, as well as for use as a scripting or glue language to connect exi
sting components together.  \n\nPython's simple, easy to learn syntax emphasi
zes readability and therefore reduces the cost of program maintenance. Python
supports modules and packages, which encourages program modularity and code r
euse. \n\nThe Python interpreter and the extensive standard library are avail
able in source or binary form without charge for all major platforms, and can
be freely distributed.\n\nOften, programmers fall in love with Python because
of the increased productivity it provides. Since there is no compilation ste
p, the edit-test-debug cycle is incredibly fast. \nDebugging Python programs
is easy: a bug or bad input will never cause a segmentation fault.\n\nInstea
d, when the interpreter discovers an error, it raises an exception. When the
program doesn't catch the exception, the interpreter prints a stack trace. A
source level debugger allows inspection of local and global variables, evalua
tion of arbitrary expressions, setting breakpoints, stepping through the code
a line at a time, and so on. \n\nThe debugger is written in Python itself, te
stifying to Python's introspective power. On the other hand, often the quicke
st way to debug a program is to add a few print statements to the source: the
fast edit-test-debug cycle makes this simple approach very effective."
```

In [126]:
```
fileobj.seek(0)
fileobj.read(16) # Return the first 16 characters of the file
```

Out[126]:
```
'Python is an int'
```

In [127]:
```
fileobj.tell() # Get the file cursor position
```

Out[127]:
```
16
```

In [128]:
```
fileobj.seek(0)
print(fileobj.readline()) # Read first line of a file.
print(fileobj.readline()) # Read second line of a file.
print(fileobj.readline()) # Read third line of a file.
```

```
Python is an interpreted, object-oriented, high-level programming language wi
th dynamic semantics. Its high-level built in data structures, combined with
dynamic typing and dynamic binding, make it very attractive for Rapid Applica
tion Development, as well as for use as a scripting or glue language to conne
ct existing components together.



Python's simple, easy to learn syntax emphasizes readability and therefore re
duces the cost of program maintenance. Python supports modules and packages,
which encourages program modularity and code reuse.
```

In [129]:
```
fileobj.seek(0)
fileobj.readlines() # Read all lines of a file.
```

Out[129]:
```
['Python is an interpreted, object-oriented, high-level programming language
with dynamic semantics. Its high-level built in data structures, combined wit
h dynamic typing and dynamic binding, make it very attractive for Rapid Appli
cation Development, as well as for use as a scripting or glue language to con
nect existing components together.  \n',
 '\n',
 "Python's simple, easy to learn syntax emphasizes readability and therefore
reduces the cost of program maintenance. Python supports modules and package
s, which encourages program modularity and code reuse. \n",
 '\n',
 'The Python interpreter and the extensive standard library are available in
source or binary form without charge for all major platforms, and can be free
ly distributed.\n',
 '\n',
 'Often, programmers fall in love with Python because of the increased produc
tivity it provides. Since there is no compilation step, the edit-test-debug c
ycle is incredibly fast. \n',
 'Debugging Python programs is easy: a bug or bad input will never cause a se
gmentation fault.\n',
 '\n',
 "Instead, when the interpreter discovers an error, it raises an exception. W
hen the program doesn't catch the exception, the interpreter prints a stack t
race. A source level debugger allows inspection of local and global variable
s, evaluation of arbitrary expressions, setting breakpoints, stepping through
the code a line at a time, and so on. \n",
 '\n',
 "The debugger is written in Python itself, testifying to Python's introspect
ive power. On the other hand, often the quickest way to debug a program is to
add a few print statements to the source: the fast edit-test-debug cycle make
s this simple approach very effective."]
```

In [130]:
```
# Read first 5 lines of a file using readline()
fileobj.seek(0)
count = 0
for i in range(5):
    if (count < 5):
        print(fileobj.readline())
    else:
        break
    count+=1
```

```
Python is an interpreted, object-oriented, high-level programming language wi
th dynamic semantics. Its high-level built in data structures, combined with
dynamic typing and dynamic binding, make it very attractive for Rapid Applica
tion Development, as well as for use as a scripting or glue language to conne
ct existing components together.


Python's simple, easy to learn syntax emphasizes readability and therefore re
duces the cost of program maintenance. Python supports modules and packages,
which encourages program modularity and code reuse.


The Python interpreter and the extensive standard library are available in so
urce or binary form without charge for all major platforms, and can be freely
distributed.
```

Write File

In [131]:
```
fileobj = open('test1.txt', 'a')
fileobj.write('THIS IS THE NEW CONTENT APPENDED IN THE FILE') # Append content
fileobj.close()
fileobj = open('test1.txt')
fileobj.read()
```

Out[131]:
```
"Python is an interpreted, object-oriented, high-level programming language w
ith dynamic semantics. Its high-level built in data structures, combined with
dynamic typing and dynamic binding, make it very attractive for Rapid Applica
tion Development, as well as for use as a scripting or glue language to conne
ct existing components together.  \n\nPython's simple, easy to learn syntax e
mphasizes readability and therefore reduces the cost of program maintenance.
Python supports modules and packages, which encourages program modularity and
code reuse. \n\nThe Python interpreter and the extensive standard library are
available in source or binary form without charge for all major platforms, an
d can be freely distributed.\n\nOften, programmers fall in love with Python b
ecause of the increased productivity it provides. Since there is no compilati
on step, the edit-test-debug cycle is incredibly fast. \nDebugging Python pro
grams is easy: a bug or bad input will never cause a segmentation fault.\n\nI
nstead, when the interpreter discovers an error, it raises an exception. When
the program doesn't catch the exception, the interpreter prints a stack trac
e. A source level debugger allows inspection of local and global variables, e
valuation of arbitrary expressions, setting breakpoints, stepping through the
code a line at a time, and so on. \n\nThe debugger is written in Python itsel
f, testifying to Python's introspective power. On the other hand, often the q
uickest way to debug a program is to add a few print statements to the sourc
e: the fast edit-test-debug cycle makes this simple approach very effective.T
HIS IS THE NEW CONTENT APPENDED IN THE FILE"
```

In [132]:
```
# Open the file "test1.txt" in 'write' mode
fileobj = open("test1.txt", "w")

fileobj.write("NEW CONTENT ADDED IN THE FILE. PREVIOUS CONTENT HAS BEEN OVERWR

fileobj.close()

fileobj = open('test1.txt')

file_contents = fileobj.read()

fileobj.close()


print(file_contents)
```

```
NEW CONTENT ADDED IN THE FILE. PREVIOUS CONTENT HAS BEEN OVERWRITTEN
```

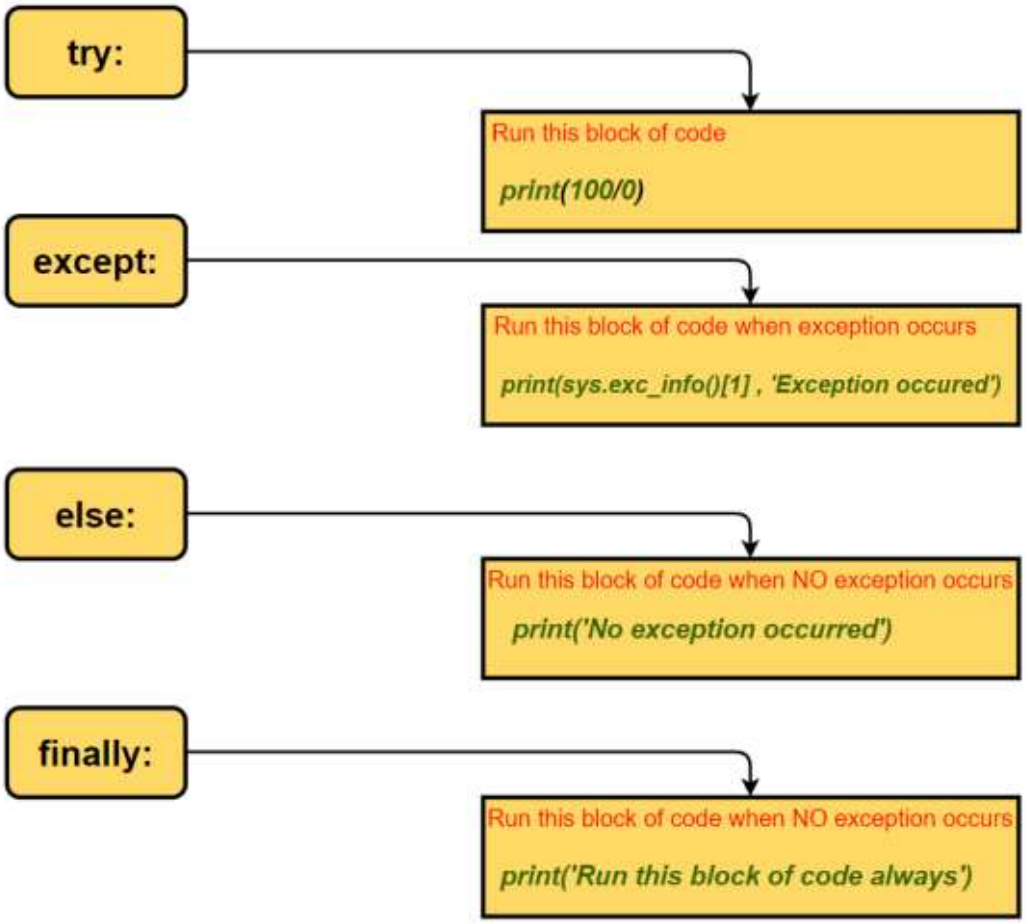
In [ ]:

In [133]:
```
fileobj = open("test2.txt", "w") # Create a new file
fileobj.write("First Line\n")
fileobj.write("Second Line\n")
fileobj.write("Third Line\n")
fileobj.write("Fourth Line\n")
fileobj.write("Fifth Line\n")
fileobj.close()
fileobj = open('test2.txt')
fileobj.readlines()
```

Out[133]:
```
['First Line\n',
 'Second Line\n',
 'Third Line\n',
 'Fourth Line\n',
 'Fifth Line\n']
```

Error & Exception Handling

- Python has many built-in exceptions (ArithmeticError, ZeroDivisionError, EOFError, IndexError, KeyError, SyntaxError, IndentationError, FileNotFoundError etc) that are raised when your program encounters an error
- When the exception occurs Python interpreter stops the current process and passes it to the calling process until it is handled. If exception is not handled the program will crash.
- Exceptions in python can be handled using a try statement. The try block lets you test a block of code for errors.
- The block of code which can raise an exception is placed inside the try clause. The code that will handle the exceptions is written in the except clause.
- The finally code block will execute regardless of the result of the try and except blocks.
- We can also use the else keyword to define a block of code to be executed if no exceptions were raised.
- Python also allows us to create our own exceptions that can be raised from the program using the raise keyword and caught using the except clause. We can define what kind of error to raise, and the text to print to the user.



In [134]:
```
import sys  # Import the sys module

try:
    print(100/0)  # ZeroDivisionError will be encountered here
except:
    print(sys.exc_info()[1], 'Exception occurred')  # This statement will be e
else:
    print('No exception occurred')  # This will be skipped as code block insid
finally:
    print('Run this block of code always')  # This will be always executed
```

```
division by zero Exception occurred
Run this block of code always
```

In [135]:
```
try:
    print(x) # NameError exception will be encountered as variable x is not de
except:
    print('Variable x is not defined')
```

```
45
```

In [136]:
```
try:
    os.remove("test3.txt") # FileNotFoundError will be encountered as "test3.t

except: # Below statement will be executed as exception occur
    print("BELOW EXCEPTION OCCURED")
    print(sys.exc_info()[1])

else:
    print('\nNo exception occurred')
finally:
    print('\nRun this block of code always')
```

```
BELOW EXCEPTION OCCURED
[WinError 2] The system cannot find the file specified: 'test3.txt'

Run this block of code always
```

In [137]:
```
import os  # Import the os module

try:
    x = int(input('Enter first number: '))
    y = int(input('Enter second number: '))
    print(x / y)
    os.remove("test3.txt")
except NameError:
    print('NameError exception occurred')
except FileNotFoundError:
    print('FileNotFoundError exception occurred')
except ZeroDivisionError:
    print('ZeroDivisionError exception occurred')
finally:
    print('This block always executes')
```

```
Enter first number: 12
Enter second number: 23
0.5217391304347826
FileNotFoundError exception occurred
This block always executes
```

In [138]:
```
# Handling specific exceptions
try:
    x = int(input('Enter first number :- '))
    y = int(input('Enter first number :- ')) # If the input entered is zero th
    print(x/y)
    os.remove("test3.txt")
except NameError:
    print('NameError exception occurred')
except FileNotFoundError:
    print('FileNotFoundError exception occurred')

except ZeroDivisionError:
    print('ZeroDivisionError exception occurred')
```

```
Enter first number :- 12
Enter first number :- 45
0.26666666666666666
FileNotFoundError exception occurred
```

In [139]:
```
try:
    x = int(input('Enter first number :- '))
    if x > 50:
        raise ValueError(x) # If value of x is greater than 50 ValueError exce
except:
    print(sys.exc_info()[0])
```

```
Enter first number :- 22
```

Built-in Exceptions

In [140]:
```
# OverflowError - This exception is raised when the result of a numeric calcul
try:
    import math
    print(math.exp(1000))
except OverflowError:
    print (sys.exc_info())
else:
    print ("Success, no error!")
```

```
(<class 'OverflowError'>, OverflowError('math range error'), <traceback objec
t at 0x000002653E633100>)
```

In [141]:
```
# ZeroDivisionError - This exception is raised when the second operator in a d
try:
    x = int(input('Enter first number :- '))
    y = int(input('Enter first number :- '))
    print(x/y)

except ZeroDivisionError:
    print('ZeroDivisionError exception occurred')
```

```
Enter first number :- 15
Enter first number :- 16
0.9375
```

In [142]:
```
# NameError - This exception is raised when a variable does not exist
try:
    print(x1)
except NameError:
    print('NameError exception occurred')
```

```
NameError exception occurred
```

In [143]:
```
# AssertionError - This exception is raised when an assert statement fails
try:
    a = 50
    b = "Raj"
    assert a == b
except AssertionError:
    print ("Assertion Exception Raised.")
```

```
Assertion Exception Raised.
```

In [144]:
```
# ModuleNotFoundError - This exception is raised when an imported module does
try:
    import MyModule
except ModuleNotFoundError:
    print ("ModuleNotFoundError Exception Raised.")
```

```
ModuleNotFoundError Exception Raised.
```

In [145]:
```
# KeyError - This exception is raised when key does not exist in a dictionary
try:
    mydict = {1:'Asif', 2:'Basit', 3:'Michael'}
    print (mydict[4])
except KeyError:
    print ("KeyError Exception Raised.")
```

```
KeyError Exception Raised.
```

In [146]:
```
# IndexError - This exception is raised when an index of a sequence does not e
try:
    mylist = [1,2,3,4,5,6]
    print (mylist[10])
except IndexError:
    print ("IndexError Exception Raised.")
```

```
IndexError Exception Raised.
```

In [147]:
```
# TypeError - This exception is raised when two different datatypes are combin
try:
    a = 50
    b = "Asif"
    c = a/b
except TypeError:
    print ("TypeError Exception Raised.")
```

```
TypeError Exception Raised.
```

In [148]:
```
# AttributeError: - This exception is raised when attribute reference or assig
try:
    a = 10
    b = a.upper()
    print(b)
except AttributeError:
    print ("AttributeError Exception Raised.")
```

```
AttributeError Exception Raised.
```

In [*]:
```
try:
    x = input('Enter first number :- ')

except:
    print('ZeroDivisionError exception occurred')
```

```
Enter first number :- 45
```

END

In [ ]: