Full Stack Development
Sesi 4

PYTHON:
FILES
ERROR HANDLING
EXCEPTION

Files

Files are named locations on disk to store related information. They are used to permanently store data in a non-volatile memory (e.g. hard disk).

Since Random Access Memory (RAM) is volatile (which loses its data when the computer is turned off), we use files for future use of the data by permanently storing them.

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 the 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

Opening Files

Python has a built-in open() function to open a file. This function returns a file object, also called a handle, as it is used to read or modify the file accordingly.

```
f = open("test.txt")    # open file in current directory

f = open("C:/Hacktiv8/README.txt")  # specifying full path
```

We can specify the mode while opening a file. In mode, we specify whether we want to read r, write w or append a to the file. We can also specify if we want to open the file in text mode or binary mode.

Opening Files

Mode	Description
`r`	Opens a file for reading. (default)
`w`	Opens a file for writing. Creates a new file if it does not exist or truncates the file if it exists.
`x`	Opens a file for exclusive creation. If the file already exists, the operation fails.
`a`	Opens a file for appending at the end of the file without truncating it. Creates a new file if it does not exist.
`t`	Opens in text mode. (default)
`b`	Opens in binary mode.
`+`	Opens a file for updating (reading and writing)

HACKTIV8

Closing Files

Closing a file will free up the resources that were tied with the file. It is done using the close() method available in Python.

Python has a garbage collector to clean up unreferenced objects but we must not rely on it to close the file.

```
f = open("test.txt", encoding = 'utf-8')

# perform file operations
f.close()
```

Writing Files

In order to write into a file in Python, we need to open it in write w, append a or exclusive creation x mode.

Writing a string or sequence of bytes (for binary files) is done using the write() method. This method returns the number of characters written to the file.

```
with open("test.txt", 'w', encoding = 'utf-8') as f:
   f.write("my first file\n")
   f.write("This file\n\n")
   f.write("contains three lines\n")
```

HACKTIV8

Reading Files

We can read the text.txt file we wrote in the above section in the following way:

```
>>> f = open("test.txt",'r',encoding = 'utf-8')
>>> f.read(4)    # read the first 4 data
'This'

>>> f.read(4)    # read the next 4 data
' is '

>>> f.read()     # read in the rest till end of file
'my first file\nThis file\ncontains three lines\n'

>>> f.read()  # further reading returns empty string
''
```

HACKTIV8

Exceptions versus Syntax Errors

Syntax errors occur when the parser detects an incorrect statement. Observe the following example:

```
print( 0 / 0 ))
  File "<stdin>", line 1
    print( 0 / 0 ))
                 ^
SyntaxError: invalid syntax
```

The arrow indicates where the parser ran into the syntax error. In this example, there was one bracket too many. Remove it and run your code again:

```
print( 0 / 0 )
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: integer division or modulo by zero
```

This time, you ran into an exception error. This type of error occurs whenever syntactically correct Python code results in an error. The last line of the message indicated what type of exception error you ran into.

Raising an Exception

We can use raise to throw an exception if a condition occurs. The statement can be complemented with a custom exception.

If you want to throw an error when a certain condition occurs using raise, you could go about it like this:

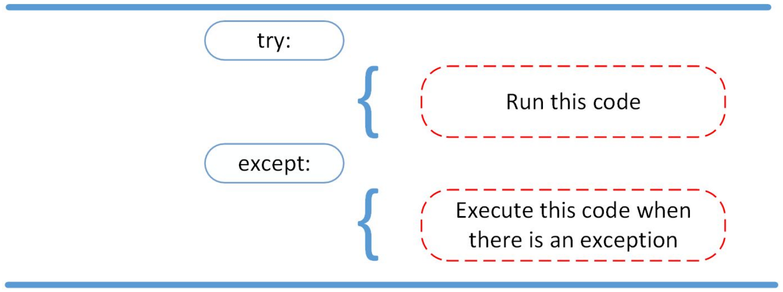
```
x = 10
if x > 5:
    raise Exception('x should not exceed 5. The value of x was: {}'.format(x))
```

try and except Block: Handling Exceptions

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

If file.log does not exist, this block of code will output the following:

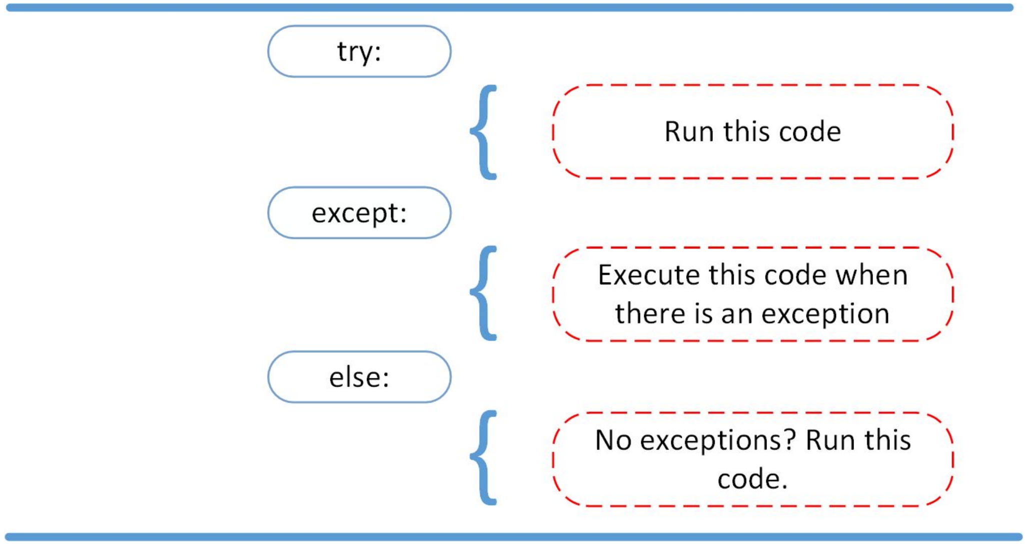
```
Could not open file.log
```



else Clause

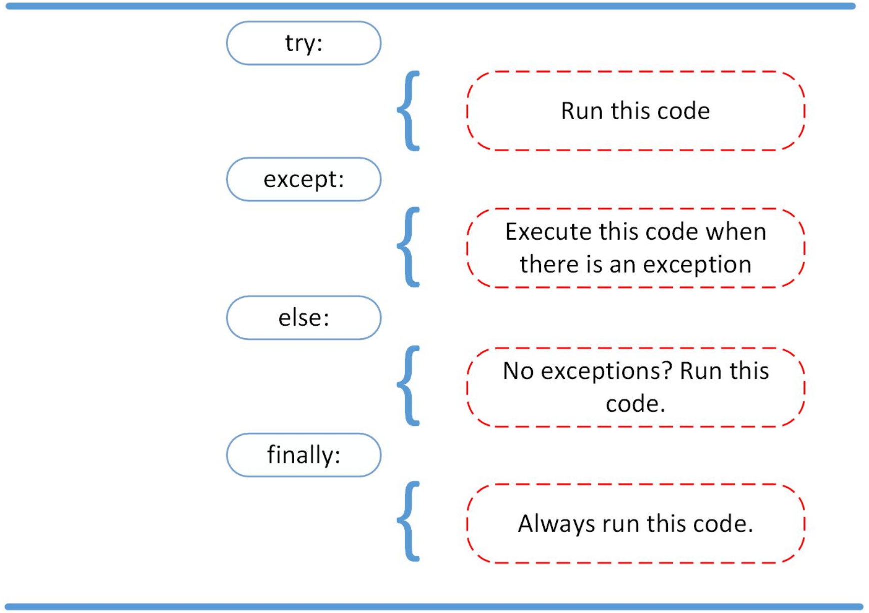
In Python, using the else statement, you can instruct a program to execute a certain block of code only in the absence of exceptions.

```
try:
    with open('file.log') as file:
        read_data = file.read()
except FileNotFoundError as error:
    print(error)
else:
    print('Executing the else clause.')
```


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

Using finally

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


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

HACKTIV8

External References

Files - [Visit Here]