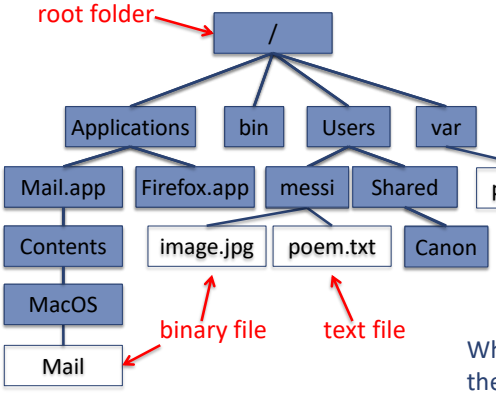
Execution Control Structures

- File Input/Output
- Errors and Exceptions

Files and the file system



The file system is the OS component that organizes files and provides a way to create, access, and modify files

Files are organized into a tree structure

- folders (or directories)
- regular files

While every file and folder has a name, it is the file pathname that identifies the file

Absolute pathnames

- `/var/poem.txt`
- `/Users/messi/poem.txt`
- `/Applications/Mail.app/`

Relative pathnames (relative to current working directory `Users`)

- `messi/poem.txt`
- `messi/Shared/image.jpg`

Opening and closing a file

Processing a file consists of:

1. Opening the file
2. Reading from and/or writing to the file
3. Closing the file

File mode `'r'` is used to open a file for reading (rather than, say, writing)

Built-in function `open()` is used to open a file

- The first input argument is the file pathname, whether absolute or relative with respect to the current working directory
- The second (optional) argument is the file mode
- Returns a "file" object

A "file" object is of a type that supports several "file" methods, including method `close()` that closes the file

```
>>> infile = open('sample.txt')
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    infile = open('sample.txt')
IOError: [Errno 2] No such file or directory:
'sample.txt'
>>> infile = open('example.txt', 'r')
>>> infile.close()
>>>
```


Open file mode

The file mode defines how the file will be accessed

Mode	Description
`r`	Reading (default)
`w`	Writing (if file exists, content is wiped)
`a`	Append (if file exists, writes are appended)
`r+`	Reading and Writing
`t`	Text (default)
`b`	Binary

These are all equivalent →

```
>>> infile = open('example.txt', 'rt')
>>> infile = open('example.txt', 'r')
>>> infile = open('example.txt', 't')
>>> infile = open('example.txt')
```

File methods

There are several "file" types; they all support similar "file" methods

- Methods `read()` and `readline()` return the characters read as a string
- Methods `readlines()` returns the characters read as a list of lines
- Method `write()` returns the number of characters written

Usage	Description
`infile.read(n)`	Read `n` characters starting from cursor; if fewer than `n` characters remain, read until the end of file
`infile.read()`	Read starting from cursor up to the end of the file
`infile.readline()`	Read starting from cursor up to, and including, the end of line character
`infile.readlines()`	Read starting from cursor up to the end of the file and return list of lines
`outfile.write(s)`	Write string `s` to file `outfile` starting from cursor
`infile.close(n)`	Close file `infile`

Reading a file

```
1 The 3 lines in this file end with the new line character.\n
2 \n
3 There is a blank line above this line.\n
```

example.txt

When the file is opened, a cursor is associated with the opened file

The initial position of the cursor is:

- at the beginning of the file, if file mode is `r`
- at the end of the file, if file mode is `a` or `w`

```
>>> infile = open('example.txt')
>>>
```

Reading a file

```
1 The 3 lines in this file end with the new line character.\n
2 \n
3 There is a blank line above this line.\n
```

example.txt

When the file is opened, a cursor is associated with the opened file

The initial position of the cursor is:

- at the beginning of the file, if file mode is `r`
- at the end of the file, if file mode is `a` or `w`

```
>>> infile = open('example.txt')
>>> infile.read(1)
'T'
>>>
```


Reading a file

```
1 The 3 lines in this file end with the new line character.\n
2 \n
3 There is a blank line above this line.\n
```

example.txt

When the file is opened, a cursor is associated with the opened file

The initial position of the cursor is:

- at the beginning of the file, if file mode is `r`
- at the end of the file, if file mode is `a` or `w`

```
>>> infile = open('example.txt')
>>> infile.read(1)
'T'
>>> infile.read(5)
'he 3 '
>>>
```

Reading a file

```
1 The 3 lines in this file end with the new line character.\n
2 \n
3 There is a blank line above this line.\n
```

example.txt

When the file is opened, a cursor is associated with the opened file

The initial position of the cursor is:

- at the beginning of the file, if file mode is `r`
- at the end of the file, if file mode is `a` or `w`

```
>>> infile = open('example.txt')
>>> infile.read(1)
'T'
>>> infile.read(5)
'he 3 '
>>> infile.readline()
'lines in this file end with the new line
character.\n'
>>>
```


Reading a file

```
1 The 3 lines in this file end with the new line character.\n
2 \n
3 There is a blank line above this line.\n
```

example.txt

When the file is opened, a cursor is associated with the opened file

The initial position of the cursor is:

- at the beginning of the file, if file mode is `r`
- at the end of the file, if file mode is `a` or `w`

```
>>> infile = open('example.txt')
>>> infile.read(1)
'T'
>>> infile.read(5)
'he 3 '
>>> infile.readline()
'lines in this file end with the new line
character.\n'
>>> infile.read()
'\nThere is a blank line above this line.\n'
>>> infile.close()
>>>
```

Patterns for reading a text file

Common patterns for reading a file:

1. Read the file content into a string
2. Read the file content into a list of words
3. Read the file content into a list of lines

Example:

```
def numChars(filename):
    'returns the number of characters in file filename'

    infile = open(filename, 'r')
    content = infile.read()
    infile.close()

    return len(content)
```


Patterns for reading a text file

Common patterns for reading a file:

1. Read the file content into a string
2. Read the file content into a list of words
3. Read the file content into a list of lines

Example:

```
def numWords(filename):
    'returns the number of words in file filename'

    infile = open(filename)
    content = infile.read()
    infile.close()
    wordList = content.split()

    return len(wordList)
```

Patterns for reading a text file

Common patterns for reading a file:

1. Read the file content into a string
2. Read the file content into a list of words
3. Read the file content into a list of lines

Example:

```
def numLines(filename):
    'returns the number of lines in file filename'

    infile = open(filename, 'r')
    lineList = infile.readlines()
    infile.close()

    return len(lineList)
```


Writing to a text file

```
1 T
2
3
4
```

test.txt

```
>>> outfile = open('test.txt', 'w')
>>> outfile.write('T')
1
>>>
```

Writing to a text file

```
1 This is the first line.
2
3
4
```

test.txt

```
>>> outfile = open('test.txt', 'w')
>>> outfile.write('T')
1
>>> outfile.write('his is the first line.')
22
>>>
```


Writing to a text file

```
1 This is the first line. Still the first line…\n
2
3
4
```

test.txt

```
>>> outfile = open('test.txt', 'w')
>>> outfile.write('T')
1
>>> outfile.write('his is the first line.')
22
>>> outfile.write(' Still the first line...\n')
25
>>>
```

Writing to a text file

```
1 This is the first line. Still the first line…\n
2 Now we are in the second line.\n
3
4
```

test.txt

```
>>> outfile = open('test.txt', 'w')
>>> outfile.write('T')
1
>>> outfile.write('his is the first line.')
22
>>> outfile.write(' Still the first line...\n')
25
>>> outfile.write('Now we are in the second line.\n')
31
>>>
```


Writing to a text file

```
1 This is the first line. Still the first line…\n
2 Now we are in the second line.\n
3 Non string value like 5 must be converted first.\n
4 Non string value like 5 must be converted first.\n
```

test.txt

```
>>> outfile = open('test.txt', 'w')
>>> outfile.write('T')
1
>>> outfile.write('his is the first line.')
22
>>> outfile.write(' Still the first line...\n')
25
>>> outfile.write('Now we are in the second line.\n')
31
>>> outfile.write('Non string value like '+str(5)+' must be converted first.\n')
49
>>> outfile.write('Non string value like {} must be converted first.\n'.format(5))
49
>>> outfile.close()
```

Types of errors

We have seen different types of errors during our python exposure so far

There are basically two types of errors:

- syntax errors
- erroneous state errors

```
>>> excuse = 'I'm sick'
SyntaxError: invalid syntax
>>> print(hour+':'+minute+':'+second)
Traceback (most recent call last):
  File "<pyshell#113>", line 1, in <module>
    print(hour+':'+minute+':'+second)
TypeError: unsupported operand type(s) for +:
'int' and 'str'
>>> infile = open('sample.txt')
Traceback (most recent call last):
  File "<pyshell#50>", line 1, in <module>
    infile = open('sample.txt')
IOError: [Errno 2] No such file or directory:
'sample.txt'
```


Syntax errors

Syntax errors are errors that are due to the incorrect format of a Python statement

- They occur while the statement is being translated to machine language and before it is being executed.

```
>>> (3+4]
SyntaxError: invalid syntax
>>> if x == 5
SyntaxError: invalid syntax
>>> print 'hello'
SyntaxError: invalid syntax
>>> lst = [4;5;6]
SyntaxError: invalid syntax
>>> for i in range(10):
print(i)
SyntaxError: expected an indented block
```

Erroneous state errors

The program execution gets into an erroneous state

When an error occurs, an "error" object is created

- This object has a type that is related to the *type* of error
- The object contains information about the error

```
>>> 3/0
Traceback (most recent call last):
  File "<pyshell#56>", line 1, in <module>
    3/0
ZeroDivisionError: division by zero
```


Erroneous state errors

The program execution gets into an erroneous state

When an error occurs, an "error" object is created

- This object has a type that is related to the *type* of error
- The object contains information about the error
- The default behavior is to print this information and interrupt the execution of the statement.

The "error" object is called an exception; the creation of an exception due to an error is called the raising of an exception

```
>>> lst
Traceback (most recent call last):
  File "<pyshell#57>", line 1, in <module>
    lst
NameError: name 'lst' is not defined
```

Erroneous state errors

The program execution gets into an erroneous state

When an error occurs, an "error" object is created

- This object has a type that is related to the *type* of error
- The object contains information about the error
- The default behavior is to print this information and interrupt the execution of the statement.

The "error" object is called an exception; the creation of an exception due to an error is called the raising of an exception

```
>>> lst = [12, 13, 14]
>>> lst[3]
Traceback (most recent call last):
  File "<pyshell#59>", line 1, in <module>
    lst[3]
IndexError: list index out of range
```


Erroneous state errors

The program execution gets into an erroneous state

When an error occurs, an "error" object is created

- This object has a type that is related to the *type* of error
- The object contains information about the error
- The default behavior is to print this information and interrupt the execution of the statement.

```
>>> lst * lst
Traceback (most recent call last):
  File "<pyshell#60>", line 1, in <module>
    lst * lst
TypeError: can't multiply sequence by non-int
of type 'list'
```

```
>>> int('4.5')
Traceback (most recent call last):
  File "<pyshell#61>", line 1, in <module>
    int('4.5')
ValueError: invalid literal for int() with
base 10: '4.5'
```

Exception types

Some of the built-in exception classes:

Exception	Explanation
`KeyboardInterrupt`	Raised when user hits Ctrl-C, the interrupt key
`OverflowError`	Raised when a floating-point expression evaluates to a value that is too large
`ZeroDivisionError`	Raised when attempting to divide by 0
`IOError`	Raised when an I/O operation fails for an I/O-related reason
`IndexError`	Raised when a sequence index is outside the range of valid indexes
`NameError`	Raised when attempting to evaluate an unassigned identifier (name)
`TypeError`	Raised when an operation of function is applied to an object of the wrong type
`ValueError`	Raised when operation or function has an argument of the right type but incorrect value