Exceptions

Syntax Error VS Exception

```
print('Welcome Back!'))
```

Syntax errors are usually caused by typos or other mistakes like an extra) here

Syntax Error VS Exception

```
print('Welcome Back!'))
```

```
/Users/sarah/Desktop/file_reader.py
  File "/Users/sarah/Desktop/file_reader.py", line 1
    print('Welcome Back!'))
                           ^
SyntaxError: unmatched ')'
```

The error output tells you where the error is.

And what the error is.

This Code Will Cause an Exception

```
acronyms = {'LOL': 'laugh out loud',
            'IDK': "I don't know",
            'TBH': 'to be honest'}

definition = acronyms['BTW']
```

Exception errors happen when the syntax is correct, but when executing the code an error occurs.

This Code Will Cause an Exception

```
acronyms = {'LOL': 'laugh out loud',
            'IDK': "I don't know",
            'TBH': 'to be honest'}

definition = acronyms['BTW']
```

```
> Traceback (most recent call last): File “file_reader.py", line 9
    definition = acronyms['BTW']
                 ~~~~~~~~^^^^^^^
  KeyError: 'BTW'
```

Again the error output tells you where the error is.

And what type of error

We Can Catch Exceptions with a `try/except` Block

```
acronyms = {'LOL': 'laugh out loud',
            'IDK': "I don't know",
            'TBH': 'to be honest'}

try:
    definition = acronyms['BTW']
except:
    print('The key does not exist')
```

Code that might cause an exception

Print an error message

```
> The key does not exist
```

The error message printed and now the program can keep running

The Format of a `try/except` Block

```
try:
    Code that might cause an exception
except:
    Print an error message

The program continues as usual...
```

Flow of Control - Getting an Exception in Our Program

```
1 acronyms = {'LOL': 'laugh out loud',
              'IDK': "I don't know",
              'TBH': 'to be honest'}

2 def = acronyms['BTW']

  print('The program keeps going...')
```

The program crashes or ends here because the Exception wasn't caught

```
> Traceback … file_reader.py", line 9, in <module>
    definition = acronyms['BTW']
                 ~~~~~~~~^^^^^^^
  KeyError: 'BTW'
```

Flow of Control - Catching an Exception in Our Program

```
1 acronyms = {'LOL': 'laugh out loud',
              'IDK': "I don't know",
              'TBH': 'to be honest'}

  try:
2     def = acronyms['BTW']
      print('Definition of ', acronym, ' is ', def)
  except:
3     print('The key ', acronym, ' does not exist')

4 print('The program keeps going...’)
```

```
> The key  BTW  does not exist
  The program keeps going...
```

Flow of Control - The Program Continues After the `try` Block

```
1 acronyms = {'LOL': 'laugh out loud',
              'IDK': "I don't know",
              'TBH': 'to be honest'}

  try:
2     def = acronyms['LOL']
3     print('Definition of ', acronym, ' is ', def)
  except:
      print('The key ', acronym, ' does not exist')

4 print('The program keeps going...’)
```

```
> Definition of LOL is laugh out loud
  The program keeps going...
```

Adding a finally

```
1  acronyms = {'LOL': 'laugh out loud',
               'IDK': "I don't know",
               'TBH': 'to be honest'}

   try:
2      def = acronyms['BTW']
       print('Definition of ', acronym, ' is ', def)
   except:
3      print('The key ', acronym, ' does not exist')
   finally:
4      print('The acronyms we have defined are:')
       for acronym in acronyms:
           print(acronym)
5  print('The program keeps going...')
```

```
> The key BTW  does
not exist

The acronyms we have
defined are:
LOL
IDK
TBH

The program keeps going..
```

The `finally` block will be executed no matter if the `try` block raises an error or not

Where Do Exceptions Come From?

As a Python Developer, you can raise (or throw) and exception if a condition occurs.

```
try:
    definition = acronyms['BTW']
except:
    print('The key does not exist')
```

For example, the Python developer that wrote the source code for dictionaries decided to raise a KeyError Exception when a key doesn't exist.

Create a Program Where We Can Raise an Exception

```
def remainder_division(a, b):
    result = a//b
    remainder = a%b
    print(a, '/', b, 'is', result,
          'remainder', remainder)



# Main program
remainder_division(10, 3)
```

```
> 10 / 3  is  3  remainder  1
```

Getting an Exception in Our New Program

```python
def remainder_division(a, b):
    result = a//b
    remainder = a%b
    print(a, '/', b, 'is', result,
          'remainder', remainder)


# Main program
remainder_division(10, 0)
```

```
> Traceback (most recent call last):
  File "/Desktop/PythonCourse/
division.py", line 29
    remainder_division(10, 0)
  File “/Desktop/PythonCourse/
division.py", line 24, in
remainder_division
    result = a//b
             ~^^~
ZeroDivisionError: integer division
or modulo by zero
```

Raising an Exception

```python
def remainder_division(a, b):
    if b == 0:
        raise ZeroDivisionError

    result = a//b
    remainder = a%b
    print(a, '/', b, 'is', result,
          'remainder', remainder)

# Main program
remainder_division(10, 0)
```

```
> Traceback (most recent call last):
  File “/Desktop/division.py", line 29
remainder_division(10, 0)
  File “/Desktop/division”, line 22, in
remainder_division
    raise ZeroDivisionError
  ZeroDivisionError
```

Raising a *Custom* Exception

```python
def remainder_division(a, b):
    if b == 0:
        raise Exception('Divisor cannot be 0')
    result = a//b
    remainder = a%b
    print(a, '/', b, 'is', result,
          'remainder', remainder)

# Main program
remainder_division(10, 0)
```

```
> Traceback (most recent call last):
  File "/Desktop/division.py", line 29
    remainder_division(10, 0)
  File “/Desktop/division", line 22, in
remainder_division
    raise Exception('The divisor cannot
be 0')
Exception: Divisor cannot be 0
```

Up Next:

Working with Files

Reading and Writing Files

Storing Software Acronyms in a Text File

input.txt

```
IDE - Integrated Development Environment
OOP - Object Oriented Programming
UX - User Experience
JSON - JavaScript Object Notation
FIFO - First In First Out
LIFO - Last In First Out
TDD - Test Driven Development
SaaS - Software as a Service
PaaS - Platform as a Service
IaaS - Infastructure as a Service
...
```

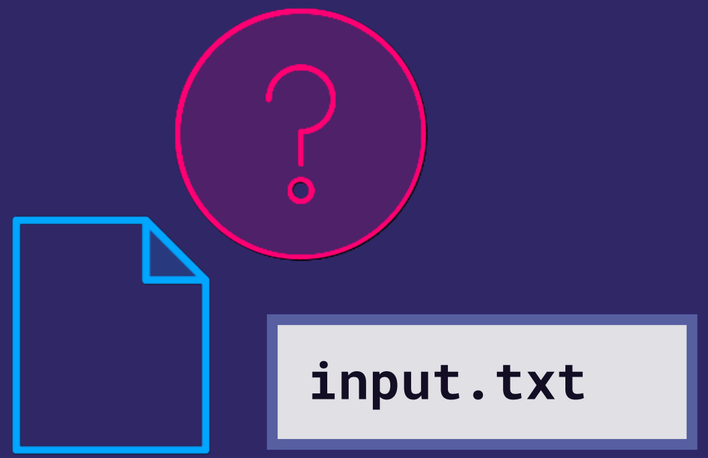
Clip of program running

Finding Software Acronyms in a Text File

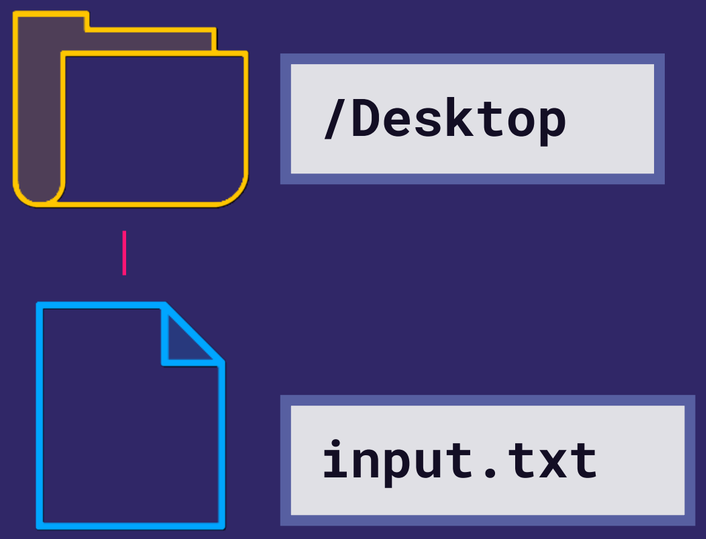
acronyms.py

1. Ask the user what acronym they want to look up?
2. Open the file
3. Read each line of the file
    4. Check if current acronym matches the user's acronym
        5. Print the definition

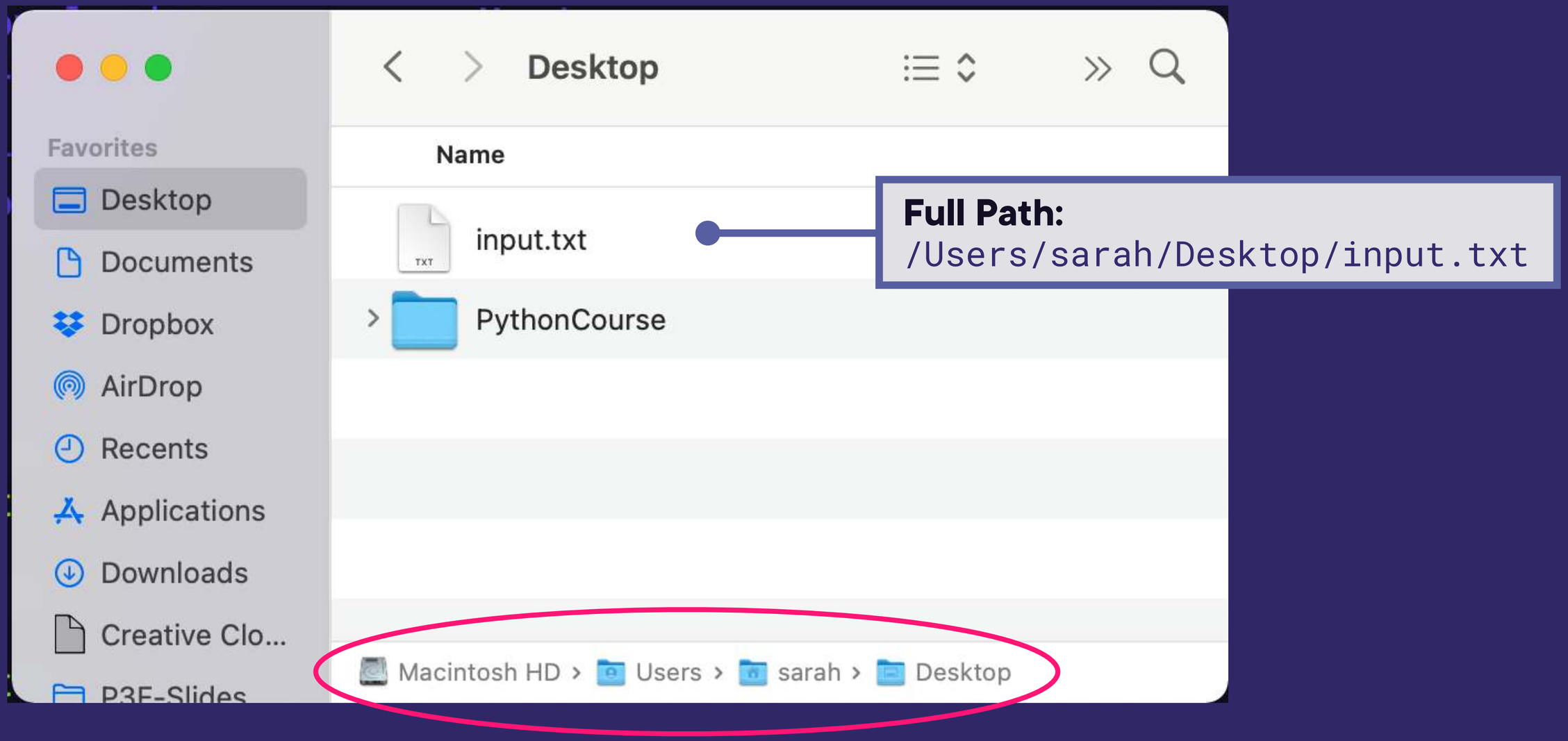
Where is our File? How to Navigate File Paths



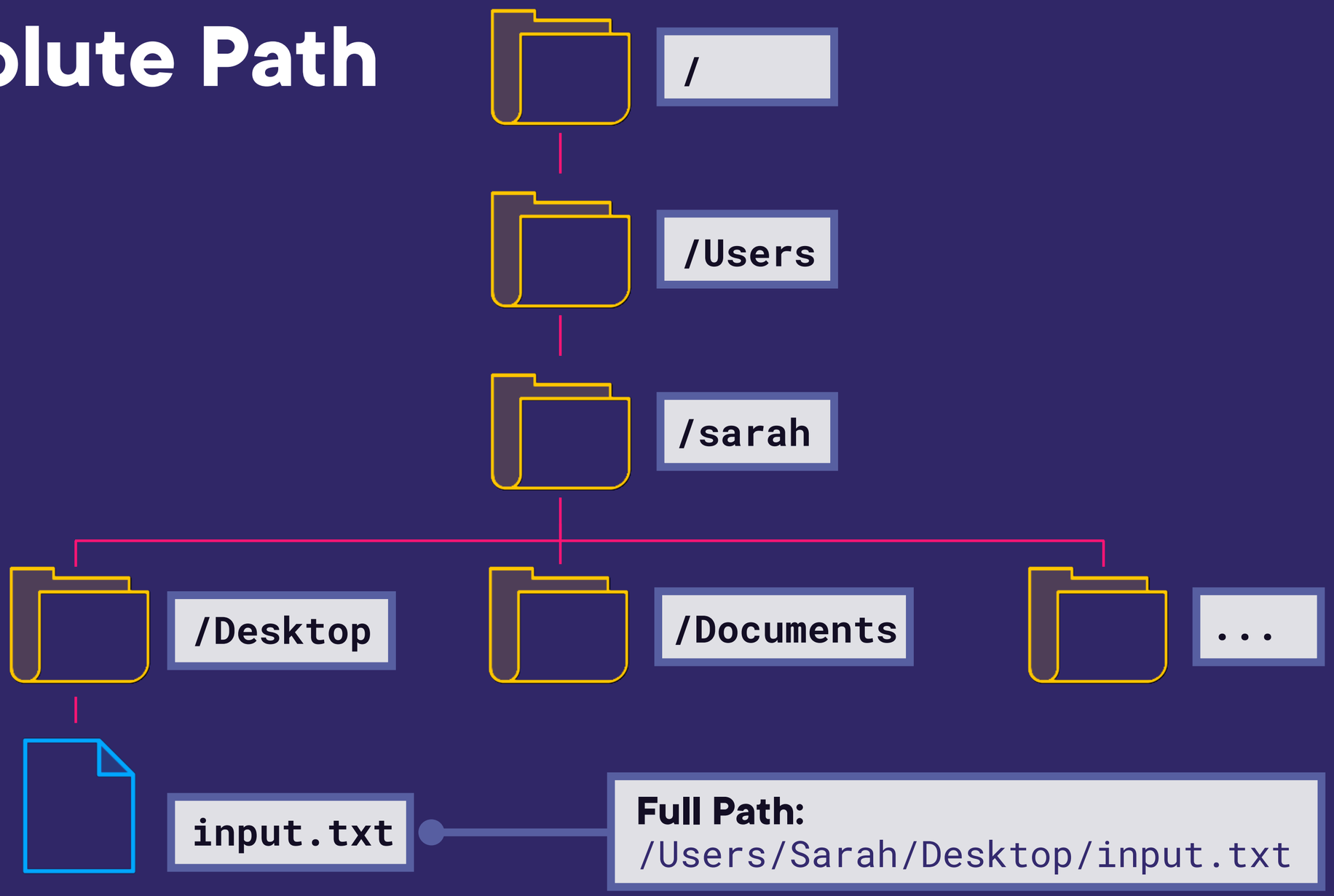
Where is our File? How to Navigate File Paths



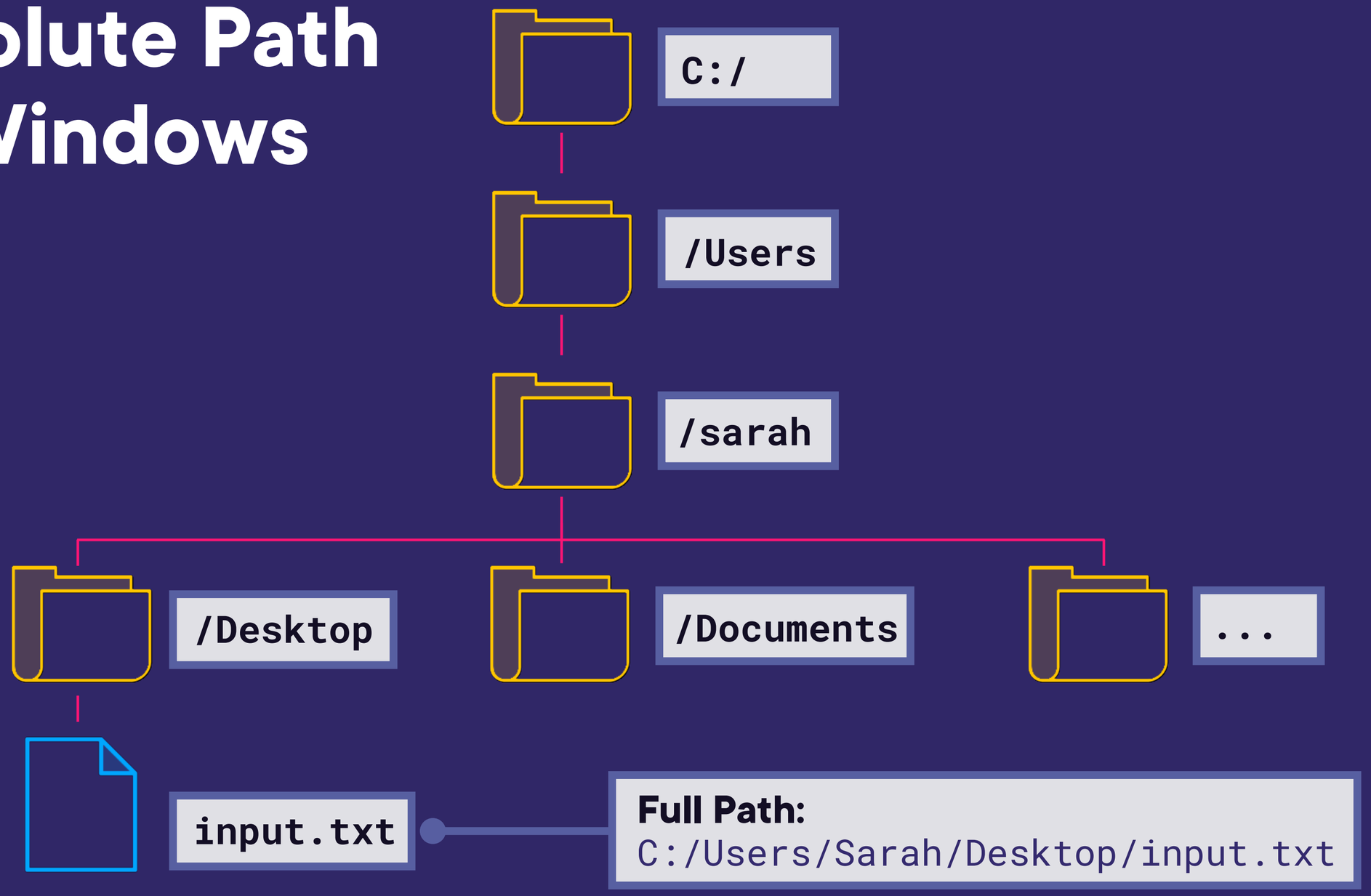
Where am I? How to Navigate File Paths



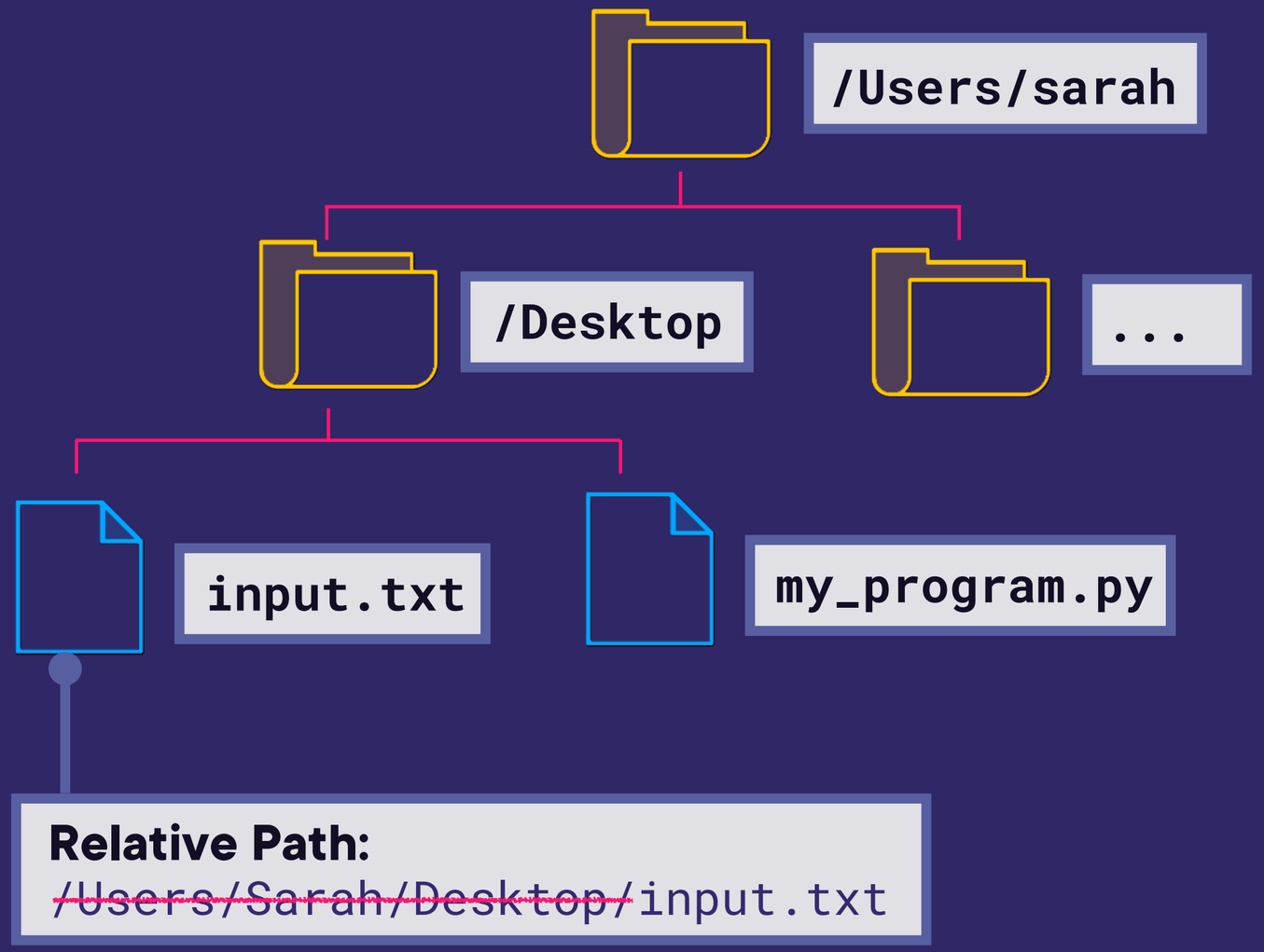
Absolute Path



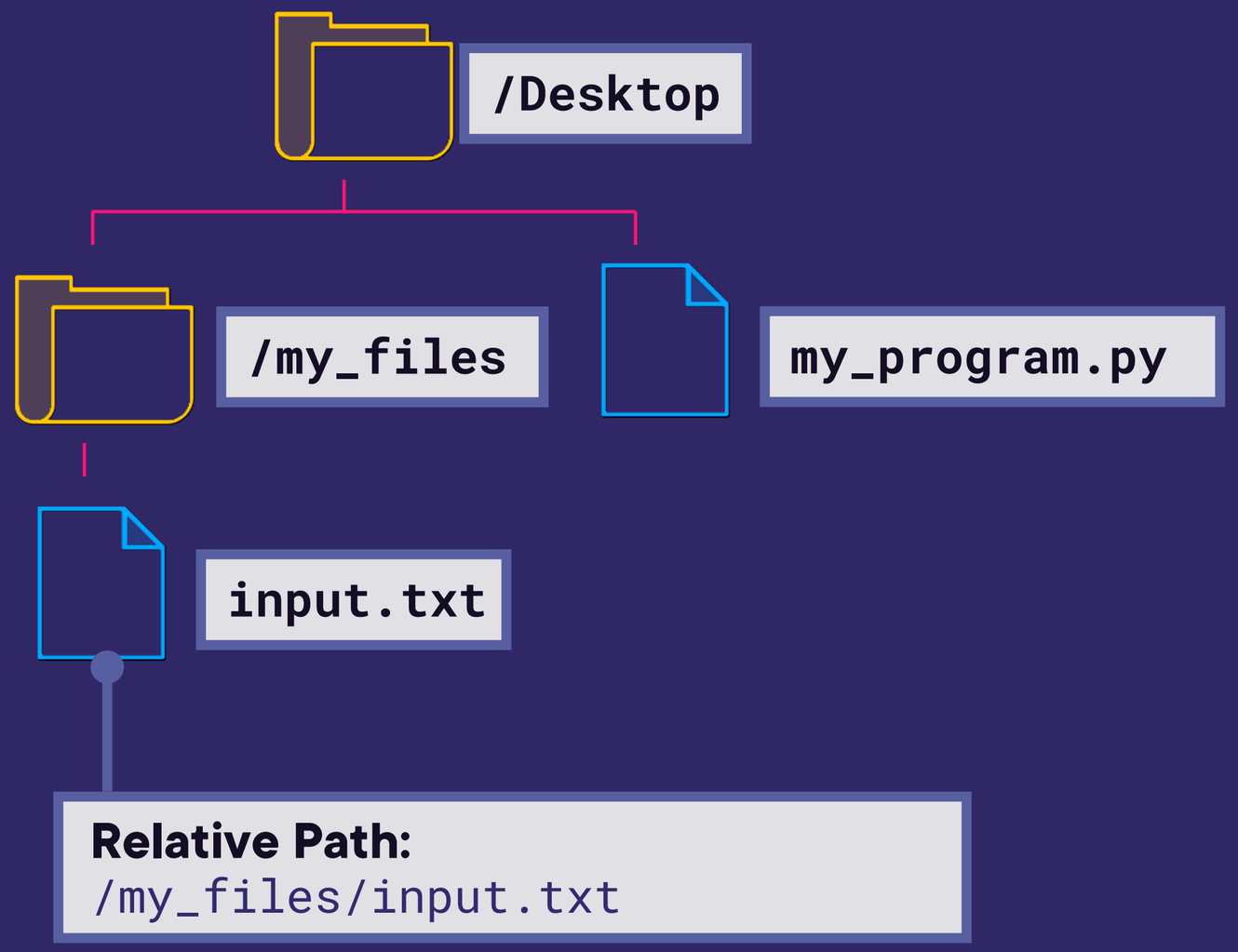
Absolute Path on Windows



Relative Path



Relative Path



Finding Acronyms in a Text File

acronyms.py

```
Ask the user what acronym they want to look up?

Open the file

Read each line of the file
    Check if current acronym matches the user's acronym
        Print the definition
```

Opening a File in Python

acronyms.py

```
file = open('acronyms.txt')
```

`open()` ***returns a*** `File` ***object that has methods like*** `read()` ***and*** `write()`

Opening a File in Python

acronyms.py

```
file = open('acronyms.txt')
```

It's very important to `close()` *a* `File` *object that has been opened...*

Opening a File in Python Using the Keyword `with`

acronyms.py

```
with open('acronyms.txt') as file:
    # Do file operations here
```

The `with` *keyword makes sure the* `File` *is properly closed when the file operations are done even if an exception is raised.*

A Longer Way to Close a File Without the Keyword `with`

acronyms.py

```
file = open('acronyms.txt')
try:
    # Do file operations here
    pass
finally:
    file.close()
```

The `finally` *block makes sure the* `File` *is properly closed when the file operations are done even if an exception is raised.*

Methods for Reading from a File Object — `read()`

acronyms.py

```
with open('acronyms.txt') as file:
    result = file.read()
    print(result)
```

The `read()` *method returns the whole file as a* `String` *by default. Or it will return the specified number of bytes.*

```
> python3 greeting.py
IDE - Integrated Development
Environment
OOP - Object Oriented Programming
UX - User Experience
JSON - JavaScript Object Notation
FIFO - First In First Out
LIFO - Last In First Out
TDD - Test Driven Development
SaaS - Software as a Service
PaaS - Platform as a Service
IaaS - Infastructure as a Service
```

Methods for Reading from a File Object — `readline()`

acronyms.py

```
with open('acronyms.txt') as file:
    result = file.readline()
    print(result)

    result = file.readline()
    print(result)
```

The `readline()` *method returns the next line of the file as a* `String`.

```
> python3 acronyms.py

IDE - Integrated Development
Environment

OOP - Object Oriented
Programming
```

Methods for Reading from a File Object — `readlines()`

acronyms.py

```
with open('acronyms.txt') as file:
    result = file.readlines()
    for line in result:
        print(line)
```

The `readlines()` *method returns a list of* `Strings` *of all of the lines in the file.*

We can loop over this list and print each line.

```
> python3 acronyms.py

IDE - Integrated Development
Environment

OOP - Object Oriented Programming

UX - User Experience

JSON - JavaScript Object Notation

FIFO - First In First Out
...
```

Using a Loop to Read from a File Object

acronyms.py

```
with open('acronyms.txt') as file:
    result = file.readlines()
    for line in result: file:
        print(line)
```

Since this type of loop is used so often there is a shortcut, we can just loop over the File Object.

Using a Loop to Read from a File Object

acronyms.py

```
with open('acronyms.txt') as file:

    for line in file:
        print(line)
```

Since this type of loop is used so often there is a shortcut, we can just loop over the File Object.

```
> python3 acronyms.py

IDE - Integrated Development
Environment

OOP - Object Oriented Programming

UX - User Experience

JSON - JavaScript Object Notation

FIFO - First In First Out
...
```

Finishing Our Program

acronyms.py

```
Ask the user what acronym they want to look up?

Open the file

Read each line of the file
    Check if current acronym matches the user's acronym
        Print the definition
```

Finishing Our Program

acronyms.py

```
look_up = input("What software acronym would you like to look up?\n")

with open('acronyms.txt') as file:
    for line in file:
        if look_up in line:
            print(line)
```

```
> python3 acronyms.py

What software acronym would you like to look up?
FIFO
First In First Out
```

Finishing Our Program

acronyms.py

```
look_up = input("What software acronym would you like to look up?\n")

with open('acronyms.txt') as file:
    for line in file:
        if look_up in line:
            print(line)
            break
```

```
> python3 acronyms.py

What software acronym would you like to look up?
FIFO
First In First Out
```

Finishing Our Program

acronyms.py

```
look_up = input("What software acronym would you like to look up?\n")

found = False
with open('acronyms.txt') as file:
    for line in file:
        if look_up in line:
            print(line)
            found = True
            break

if not found:
    print('The acronym does not exist')
```

```
> python3 acronyms.py

What software acronym would you like to look up?
FIFO
First In First Out
```

Up Next:

Demo:
Improve Our Program & Write New Acronyms to Our File

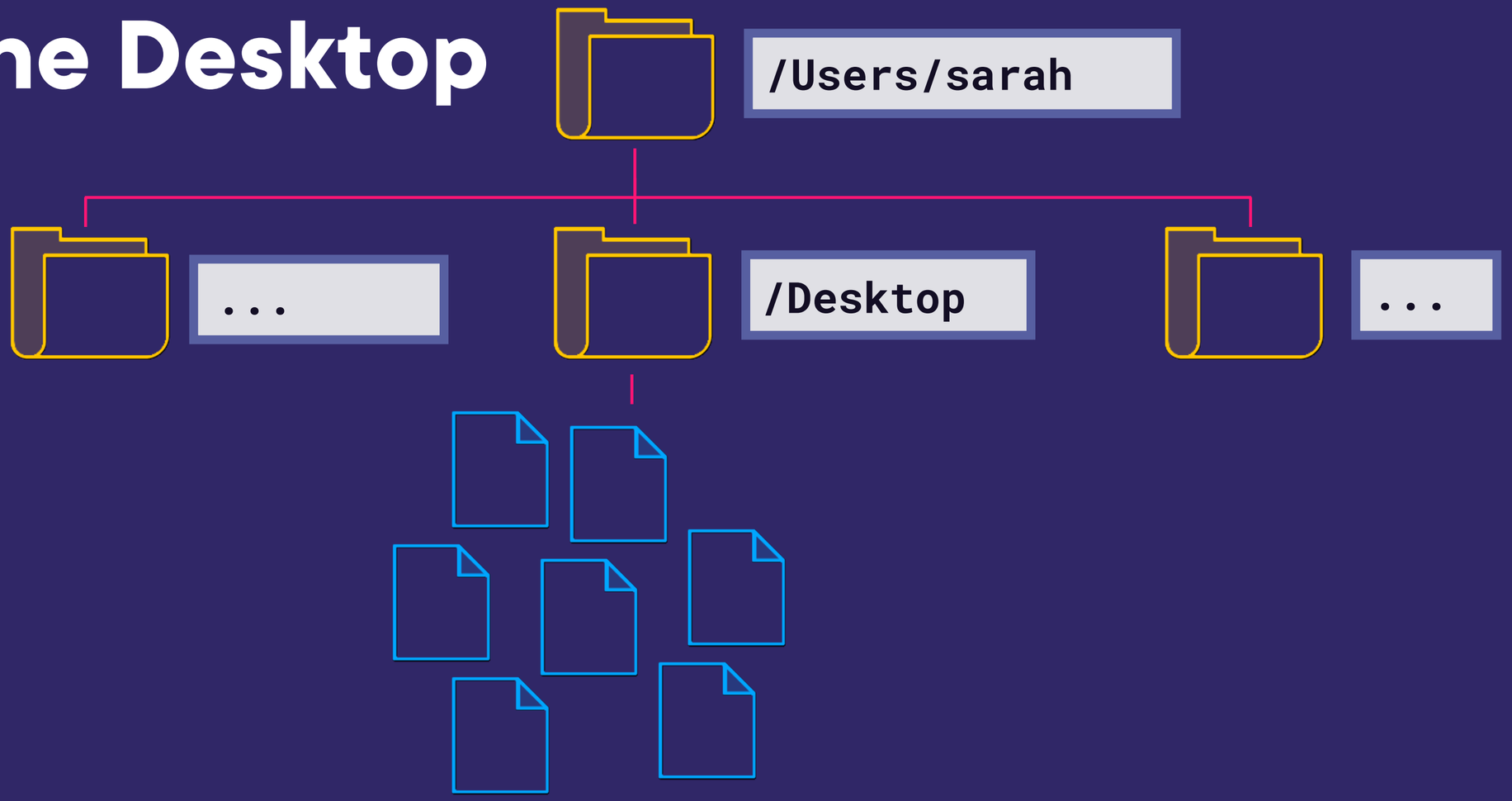
Working with Files

Overview of Working with Files in Python

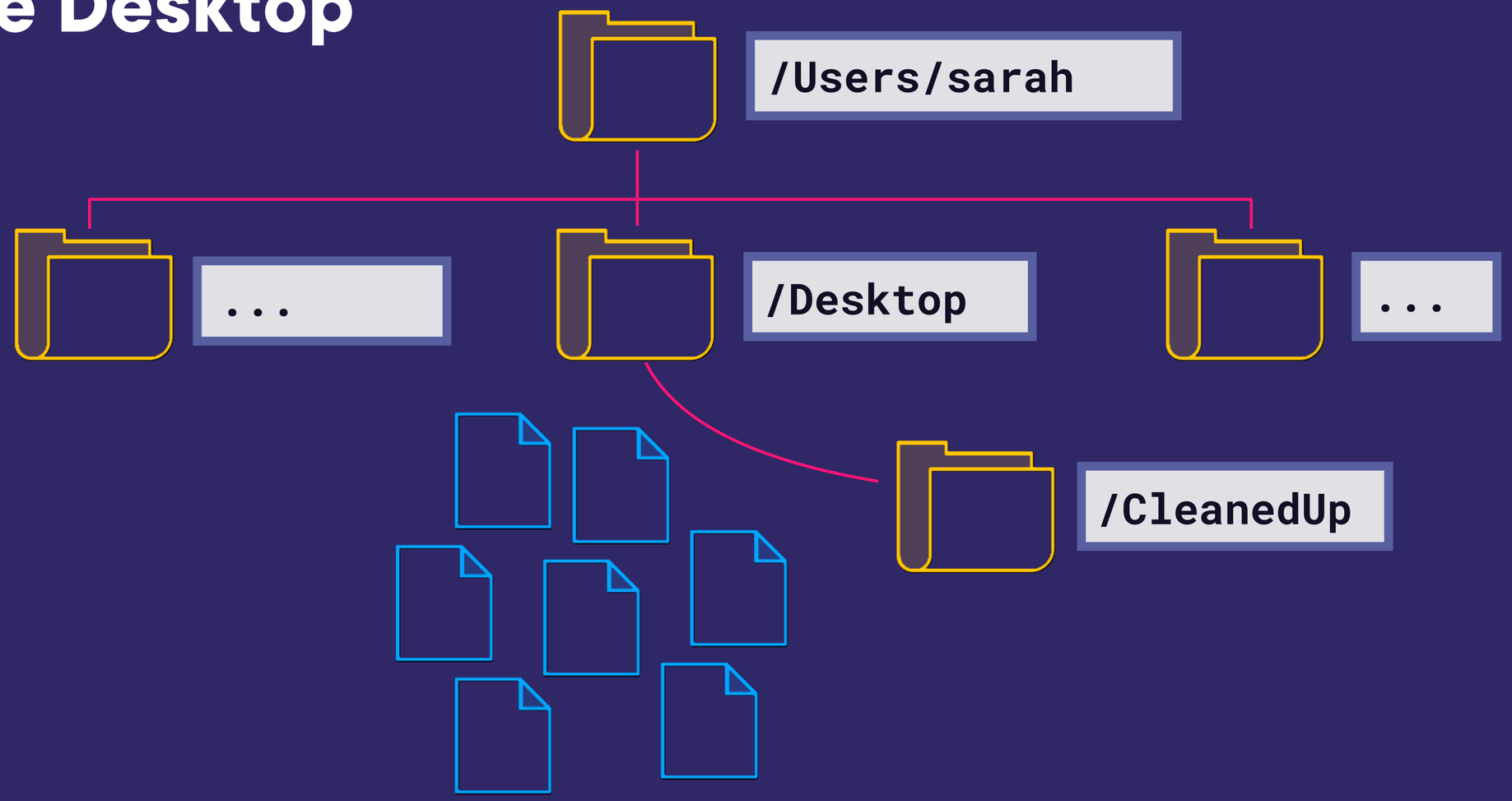
Python has several built-in modules for handling files: `os`, `shutil`, and `pathlib`

We're going to show how to use the `os` module with an example program

Too Many Files on the Desktop



Cleaning up Files on the Desktop



Cleaning Up Files on the Desktop

file_cleanup.py

1. Make the folder CleanedUp/
2. List the files in the Desktop/ folder
3. For each file in the Desktop/ folder
    4. Move the file to the CleanedUp/ folder

Making Directories

file_cleanup.py

```
import os
os.mkdir('/Users/sarah/Desktop/CleanedUp/')
```

We need to import the os *module to be able to use its functions.*

List All Entries in a Directory

file_cleanup.py

```python
import os

folder = '/Users/sarah/Desktop/'

entries = os.scandir(folder)

for entry in entries:
    print(entry.name)
```

```
> python3 file_cleanup.py

.DS_Store
PythonCourse
file_cleanup.py
acronyms.txt
temp.py
temp2.py
acronyms.py
```

Check if a Directory Entry is a File or a Subdirectory

file_cleanup.py

```python
import os

folder = '/Users/sarah/Desktop/'

entries = os.scandir(folder)

for entry in entries:
    if os.path.isfile(entry):
        print('File:', entry.name)
    elif os.path.isdir(entry):
        print('Directory:', entry.name)
```

```
> python3 file_cleanup.py

File: .DS_Store
Directory: PythonCourse
File: file_cleanup.py
File: acronyms.txt
File: temp.py
File: temp2.py
File: acronyms.py
```

Create an Absolute Path Name

file_cleanup.py

```
import os

folder_destination = '/Users/sarah/Desktop/CleanedUp/'

new_name = os.path.join(folder_destination, 'new.txt')
```

We could use string concatenation instead:

```
new_name = '/Users/sarah/Desktop/CleanedUp/' + 'new.txt'
```

But `os.path.join()` *checks for correct format and is a best practice.*

Move a File

file_cleanup.py

```
import os

folder_original = '/Users/sarah/Desktop/'
folder_destination = '/Users/sarah/Desktop/CleanedUp/'

location_original = os.path.join(folder_original,'new.txt')
location_destination = os.path.join(folder_destination, 'new.txt')

os.rename(location_original, location_destination)
```

`os.rename()` *allows us to move a file to a new path*

Cleaning Up Files on the Desktop

file_cleanup.py

1. Make the folder CleanedUp/
2. List the files in the Desktop/ folder
3. For each file in the Desktop/ folder
    4. Move the file to the CleanedUp/ folder

Cleaning Up Files on the Desktop

```
import os

folder_original = '/Users/sarah/Desktop/'
folder_destination = '/Users/sarah/Desktop/CleanedUp/'
```

Start with where we're moving our files from and to.

Cleaning Up Files on the Desktop

```
import os

folder_original = '/Users/sarah/Desktop/'
folder_destination = '/Users/sarah/Desktop/CleanedUp/'

os.mkdir(folder_destination)
```

Then we'll create the new `CleanedUp` *directory.*

Cleaning Up Files on the Desktop

```
import os

folder_original = '/Users/sarah/Desktop/'
folder_destination = '/Users/sarah/Desktop/CleanedUp/'
os.mkdir(folder_destination)

entries = os.scandir(folder_original)
```

List the entries in the Desktop folder.

Cleaning Up Files on the Desktop

```
import os

folder_original = '/Users/sarah/Desktop/'
folder_destination = '/Users/sarah/Desktop/CleanedUp/'
os.mkdir(folder_destination)

for entry in os.scandir(folder_original):
```

We can combine listing the entries with the for loop.

Cleaning Up Files on the Desktop

```
import os

folder_original = '/Users/sarah/Desktop/'
folder_destination = '/Users/sarah/Desktop/CleanedUp/'
os.mkdir(folder_destination)

for entry in os.scandir(folder_original):

    location_original = os.path.join(folder_original, entry.name)
    location_destination = os.path.join(folder_destination, entry.name)
```

In order to move the files, we need to create the paths first.

Cleaning Up Files on the Desktop

```
import os

folder_original = '/Users/sarah/Desktop/'
folder_destination = '/Users/sarah/Desktop/CleanedUp/'
os.mkdir(folder_destination)

for entry in os.scandir(folder_original):

    location_original = os.path.join(folder_original, entry.name)
    location_destination = os.path.join(folder_destination, entry.name)

    os.rename(location_original, location_destination)
```

Finally we can move the files.

Cleaning Up Files on the Desktop

```
import os

folder_original = '/Users/sarah/Desktop/'
folder_destination = '/Users/sarah/Desktop/CleanedUp/'
os.mkdir(folder_destination)

for entry in os.scandir(folder_original):
    location_original = os.path.join(folder_original, entry.name)
    location_destination = os.path.join(folder_destination, entry.name)
    if os.path.isfile(loc_original):
        os.rename(location_original, location_destination)
```

Let's also check that we're only moving files, not directories.

Up Next:

Demo:
Improve Our Program