Errors in Python:

1. **Syntax Errors**: Syntax errors, also known as parsing errors, occur when the Python interpreter is unable to understand a line of code due to invalid syntax. These errors prevent the code from executing.

Example:

```
x = 5
if x == 5
    print("x is 5")
```

2. **Indentation Errors**: Indentation errors occur when there are inconsistencies in the indentation of code blocks, such as loops, conditional statements, and function definitions.

Example:

```
for i in range(5):
print(i)
```

Exceptions in Python:

1. **Runtime Errors or Exceptions**: Unlike syntax errors, runtime errors occur during the execution of the program. These errors are called exceptions and can be handled using try-except blocks.

Example:

```
x = 5
y = 0
z = x / y  # ZeroDivisionError occurs here
```

Exception Handling in Python:

1. **try-except Blocks**: Python provides a mechanism to handle exceptions using the `try`, `except`, `else`, and `finally` blocks.

Example:

```
try:
    x = 5
    y = 0
    z = x / y
except ZeroDivisionError:
    print("Error: Division by zero!")
```

2. **Multiple Except Blocks**: You can have multiple `except` blocks to handle different types of exceptions.

Example:

try:

    # code that may raise exceptions

except ValueError:

    # handle ValueError

except ZeroDivisionError:

    # handle ZeroDivisionError

except Exception as e:

    # handle other exceptions

```
try:
    int("abc")
except ValueError:
    print("ValueError occurred")
except ZeroDivisionError:
    print("ZeroDivisionError occurred")
except Exception as e:
    print("Other exception occurred:", e)
```

3. **else Block**: The `else` block is executed if no exceptions are raised in the `try` block.

Example:

try:

    # code that may raise exceptions

except ValueError:

    # handle ValueError

else:

    # execute if no exception

```
try:
    x = 5
    y = 2
    z = x / y
except ZeroDivisionError:
    print("Error: Division by zero!")
else:
    print("Division successful, result:", z)
```

4. **finally Block**: The `finally` block is always executed, regardless of whether an exception occurred.

Example:

```
try:
    # code that may raise exceptions
except ValueError:
    # handle ValueError
finally:
    # always executed
```

```
try:
    x = 5
    y = 0
    z = x / y
except ZeroDivisionError:
    print("Error: Division by zero!")
finally:
    print("This block is always executed")
```

5. **Raising Exceptions**: You can raise exceptions manually using the `raise` statement.

Example:

```
x = -5
if x < 0:
    raise ValueError("x should be a positive number")
```

6. **Handling Multiple Exceptions**: You can handle multiple exceptions using a tuple in the `except` block.

Example:

```
try:
    # code that may raise exceptions
except (ValueError, TypeError):
    # handle ValueError or TypeError
```

```
try:
    int("abc")
except (ValueError, TypeError):
    print("Either ValueError or TypeError occurred")
```

7. **Exception Propagation**: If an exception is not handled in a function, it propagates up the call stack until it is caught by an exception handler.

Example:

```
def divide(x, y):
    return x / y

try:
    result = divide(5, 0)
except ZeroDivisionError:
    print("Error: Division by zero!")
```

A full program to demonstrate the use of exception handlings Eg. 1

```
1. def divide(x, y):
2.     try:
3.         result = x / y
4.     except ZeroDivisionError:
5.         print("Error: Division by zero!")
6.     else:
7.         print("Division successful, result:", result)
8.     finally:
9.         print("Finally block executed")
10.
11.def main():
12.    try:
13.        num1 = float(input("Enter the numerator: "))
14.        num2 = float(input("Enter the denominator: "))
15.        divide(num1, num2)
16.    except ValueError:
17.        print("Error: Invalid input. Please enter numeric values.")
18.
19.if __name__ == "__main__":
20.    main()
```

A full program to demonstrate the use of exception handlings Eg. 2

```
1. def find_element(my_list, target):
2.     try:
3.         index = my_list.index(target)
4.     except ValueError:
5.         print(f"Error: '{target}' not found in the list.")
6.         return None
7.     else:
8.         return index
9.     finally:
10.        print("Search process completed.")
11.
12.def main():
13.    my_list = [1, 3, 5, 7, 9]
14.    try:
15.        target = int(input("Enter the number to find: "))
16.        index = find_element(my_list, target)
17.        if index is not None:
18.            print(f"'{target}' found at index {index}.")
19.    except ValueError:
20.        print("Error: Please enter a valid integer.")
21.
22.if __name__ == "__main__":
23.    main()
```