**Python Exception Handling**

Exception handling in Python is a mechanism to handle runtime errors gracefully, ensuring that the program doesn't crash unexpectedly. Python provides a structured way to catch and handle exceptions using `try`, `except`, `else`, and `finally` blocks.  
  
Here’s a detailed tutorial on exception handling in Python:  
  
---  
  
## \*\*1. Basic Syntax of Exception Handling\*\*  
  
The basic structure of exception handling in Python is:  
  
```python  
try:  
 # Code that might raise an exception  
 risky\_code()  
except ExceptionType:  
 # Code to handle the exception  
 handle\_exception()  
else:  
 # Code to execute if no exception occurs  
 no\_exception\_occurred()  
finally:  
 # Code that will always execute (optional)  
 cleanup\_code()  
```  
  
---  
  
## \*\*2. Example: Handling a Specific Exception\*\*  
  
You can catch specific exceptions using the `except` block. For example:  
  
```python  
try:  
 num = int(input("Enter a number: "))  
 result = 10 / num  
 print(f"Result: {result}")  
except ZeroDivisionError:  
 print("Error: Division by zero is not allowed.")  
except ValueError:  
 print("Error: Invalid input. Please enter a number.")  
```  
  
### \*\*Explanation:\*\*  
- If the user enters `0`, a `ZeroDivisionError` is raised.  
- If the user enters a non-numeric value, a `ValueError` is raised.  
  
---  
  
## \*\*3. Catching Multiple Exceptions\*\*  
  
You can handle multiple exceptions in a single `except` block by using a tuple:  
  
```python  
try:  
 num = int(input("Enter a number: "))  
 result = 10 / num  
 print(f"Result: {result}")  
except (ZeroDivisionError, ValueError) as e:  
 print(f"Error: {e}")  
```  
  
---  
  
## \*\*4. Using the `else` Block\*\*  
  
The `else` block runs if no exception occurs in the `try` block:  
  
```python  
try:  
 num = int(input("Enter a number: "))  
 result = 10 / num  
except ZeroDivisionError:  
 print("Error: Division by zero is not allowed.")  
except ValueError:  
 print("Error: Invalid input. Please enter a number.")  
else:  
 print(f"Result: {result}")  
```  
  
---  
  
## \*\*5. The `finally` Block\*\*  
  
The `finally` block is always executed, regardless of whether an exception occurred or not. It is typically used for cleanup operations:  
  
```python  
try:  
 file = open("example.txt", "r")  
 content = file.read()  
 print(content)  
except FileNotFoundError:  
 print("Error: File not found.")  
finally:  
 print("Closing the file.")  
 file.close()  
```  
  
---  
  
## \*\*6. Raising Exceptions\*\*  
  
You can raise exceptions manually using the `raise` keyword:  
  
```python  
try:  
 age = int(input("Enter your age: "))  
 if age < 0:  
 raise ValueError("Age cannot be negative.")  
except ValueError as e:  
 print(f"Error: {e}")  
```  
  
---  
  
## \*\*7. Custom Exceptions\*\*  
  
You can define your own exceptions by creating a custom exception class that inherits from the built-in `Exception` class:  
  
```python  
class NegativeNumberError(Exception):  
 pass  
  
try:  
 num = int(input("Enter a positive number: "))  
 if num < 0:  
 raise NegativeNumberError("Negative numbers are not allowed.")  
except NegativeNumberError as e:  
 print(f"Error: {e}")  
```  
  
---  
  
## \*\*8. Best Practices for Exception Handling\*\*  
  
1. \*\*Catch Specific Exceptions:\*\* Avoid using a generic `except` block unless necessary.  
2. \*\*Avoid Silencing Exceptions:\*\* Always log or handle exceptions properly.  
3. \*\*Use `finally` for Cleanup:\*\* Ensure resources like files or database connections are closed.  
4. \*\*Don't Overuse Exceptions:\*\* Use exceptions for exceptional cases, not for normal control flow.  
  
---  
  
## \*\*9. Full Example\*\*  
  
Here’s a complete example that combines multiple concepts:  
  
```python  
class CustomError(Exception):  
 pass  
  
try:  
 num = int(input("Enter a number: "))  
 if num < 0:  
 raise CustomError("Negative numbers are not allowed.")  
 result = 10 / num  
except ZeroDivisionError:  
 print("Error: Division by zero is not allowed.")  
except ValueError:  
 print("Error: Invalid input. Please enter a number.")  
except CustomError as e:  
 print(f"Error: {e}")  
else:  
 print(f"Result: {result}")  
finally:  
 print("Execution completed.")  
```  
  
---  
  
By mastering exception handling, you can write robust and error-tolerant Python programs.