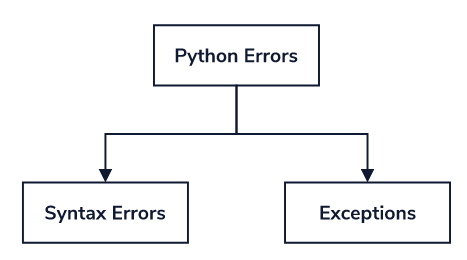
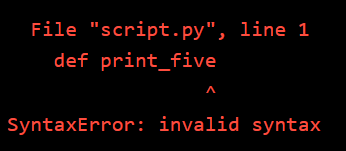
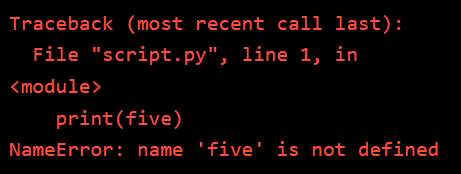
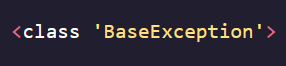
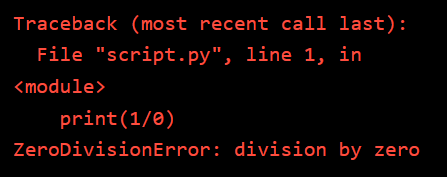
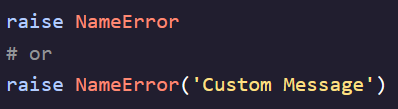
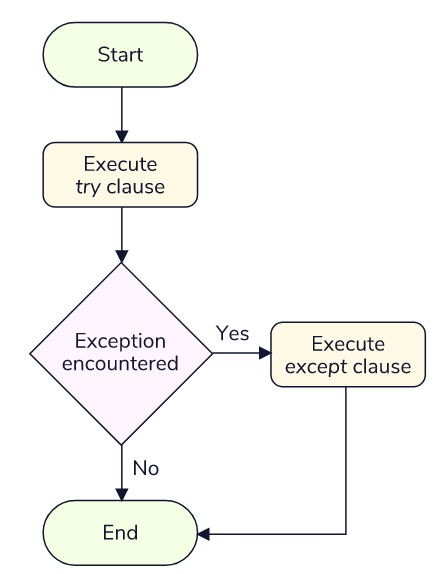
**Exceptions:**

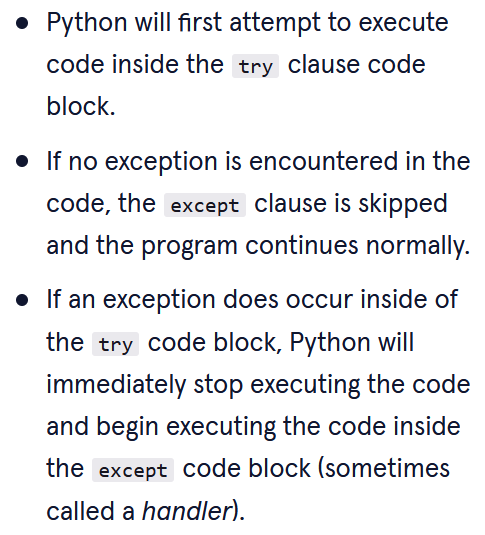
- Errors are an integral part of working with Python and it’s important to know how to control them and use them to our advantage  
- Errors fall into one of two categories: ***Syntax Errors*** and ***Exceptions***  
- ***Syntax Errors*** – Mistakes in the structure of Python code that is caught during a special parsing stage before program execution and prevent the entire program from running  
  
- ***Exceptions*** – Runtime errors that occur during program execution when code causing the error is reached. Occur even when code is syntactically correct  
  
- [Built-in-exceptions](https://docs.python.org/3/library/exceptions.html#built-in-exceptions) in Python cover naming conventions, mathematical errors, and others all the way to operating system errors  
- Exceptions are objects and inherit directly from the *BaseException* class and/or the *Exception* class  
 - Can review the class of an error by using the \_\_bases\_\_ attribute   
   
   
***- Traceback*** – A summary that includes the exception type, a message, and a series of function calls preceding the exception along with file names and line numbers   


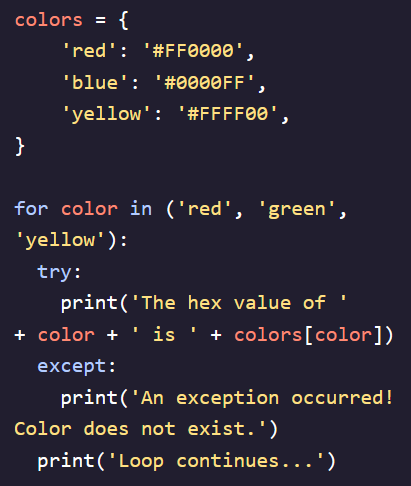
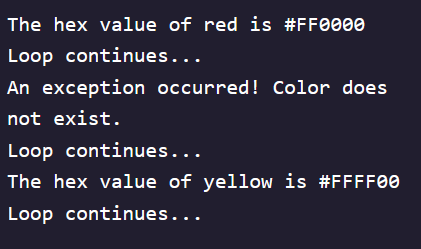
**Raising Exceptions:**

- Can use the ***raise*** keyword to thrown an exception at any time, even when there normally wouldn’t be one  
- We might want to raise an exception when we think a mistake has or will occur in our program. This lets us stop program execution immediately and provide a useful error message instead of allowing mistakes to occur that may be difficult to diagnose at a later point  
- Typically pair ***raise*** keyword with a specific exception class name or call a constructor to provide a custom error message  
- Use the exception that will provide the best explanation for the expected error for both the user and anyone that will read the code  


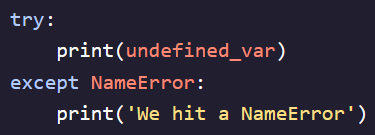
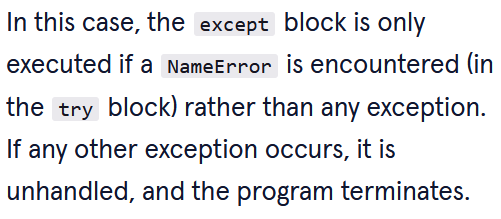
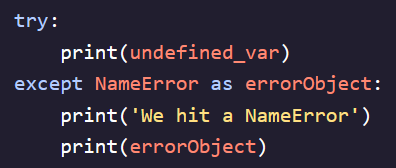
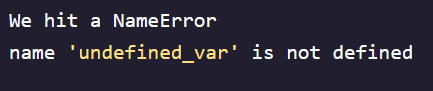
**Exception Handling:**

- It is possible for programs to continue executing even after encountering an exception, this is known as ***Exception Handling*** and is accomplished using the ***try/except*** clause  




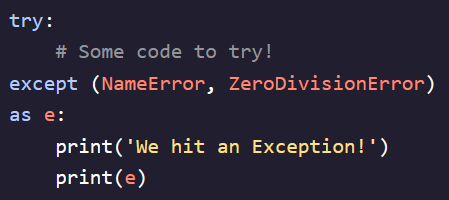
 

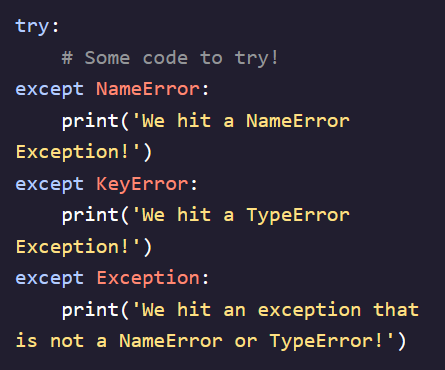
**Catching Specific Exceptions:**

- Best practice is to be as specific as possible with exceptions we want to raise unless there is a reason for catching any type of exception  
- Catch specific exceptions by listing them after the except keyword  
   
- Can capture the exception as a variable using the ***as*** keyword  
 

**Handling Multiple Exceptions:**

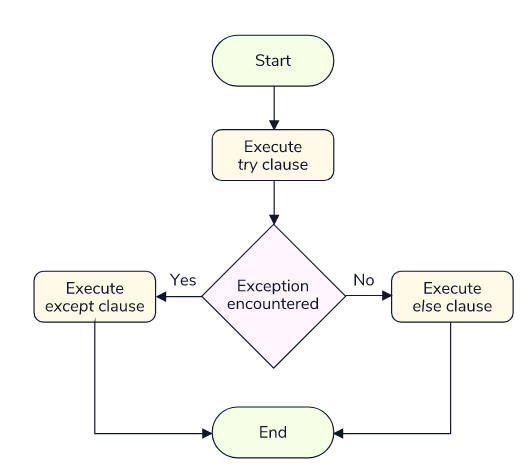
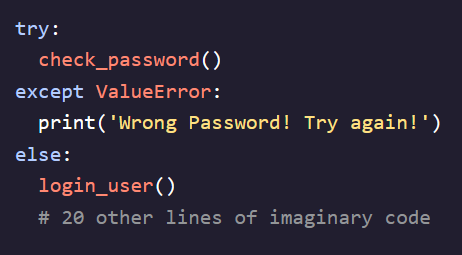
- Python allows us to handle multiple exceptions at once by listing more than one exception type in an *except* clause using a tuple



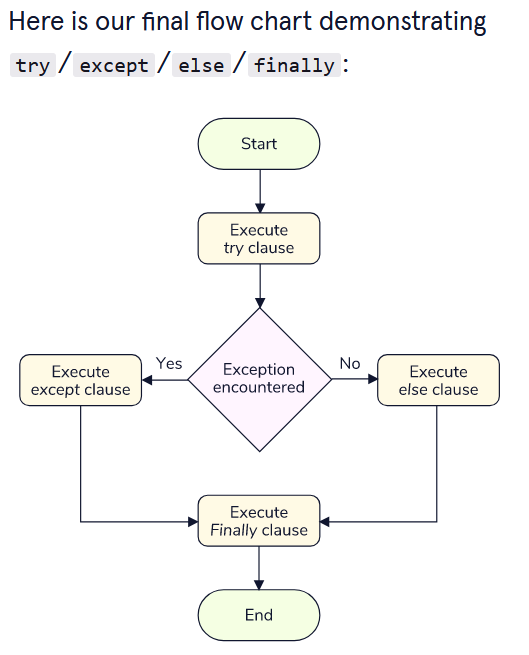
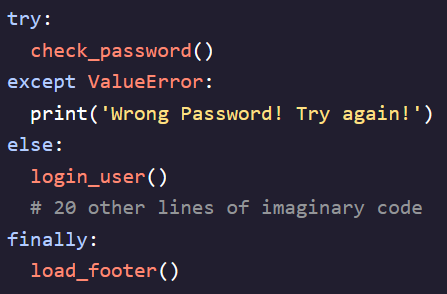
- Can also pair multiple *except* clauses with a single *try* clause to handle specific exceptions differently   


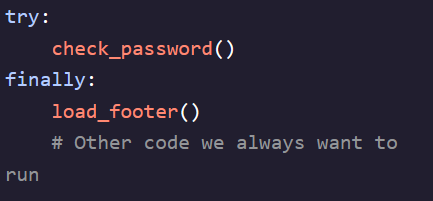
**Else Clause:**

- In addition to *except* clause, we can run an *else* clause that executes if there are **no** exceptions

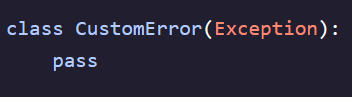
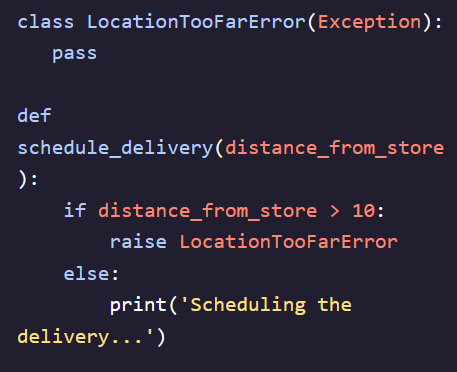
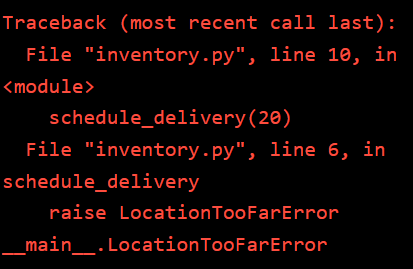
 

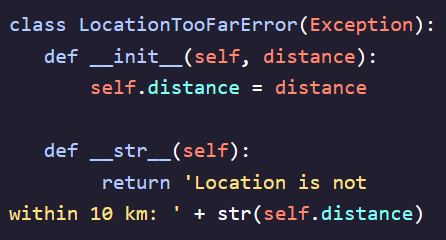
**Finally Clause:**

- Allows us to execute code regardless of whether an exception occurs  
- Happens after either the *except* or *else* clauses execute  
 

- Can also be used independently without *except* or *else* clauses   


**User-Defined Exceptions:**

- Python allows us to create custom exceptions that are specific to a program or module  
- To create a custom exception, derive a subclass from the built-in *Exception* class  
   
Returns:  


- Can customize exceptions even further by providing custom error messages  
 Returns: 