Exception Handling

- Programming Errors
- Debugging
- Exception Handling

Python Error Messages

- A key programming task is debugging when a program does not work correctly or as expected.
- If Python finds an error in your code, it raises an exception.
  e.g. We try to convert incompatible types
  e.g. We try to read a non-existent file
  Also... When we have invalid syntax in our code (a "typo")

```
number = int("UCD")
```

```
Traceback (most recent call last):
  File "test.py", line 1, in <module>
    number = int("UCD")
ValueError: invalid literal for int() with base 10: 'UCD'
```

Where the error occurred

Type of exception that has occurred

Text describing the error

Python Error Messages

```
d = {"Ireland": "Dublin"}
d["France"]
```

```
Traceback (most recent call last):
  File "test2.py", line 2, in <module>
    d["France"]
KeyError: 'France'
```

Where the error occurred

Type of exception that has occurred

```
def showuser(username):
  print(user_name)

showuser("bob")
```

```
Traceback (most recent call last):
  File "test3.py", line 4, in <module>
    showuser("bob")
  File "test3.py", line 2, in showuser
    print(user_name)
NameError: name 'user_name' is not defined
```

Error originated here

Exception Handling

- By default, an exception will terminate a script or notebook.
- We can handle errors in a structured way by "catching" exceptions. We plan in advance for errors that might occur...

General Format

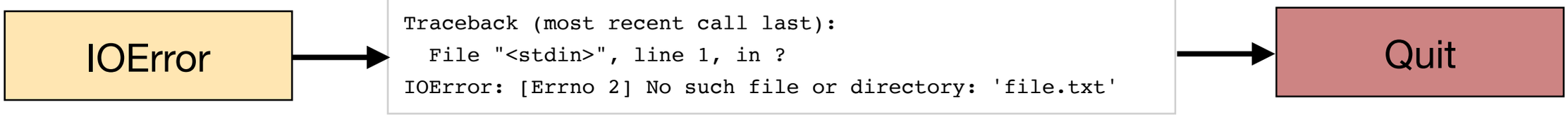
```
try:
   <block of code>
except <errorType>:
   <error handling block>
```

```
try:
    f = open("file.txt","r")
except IOError:
    print("Got error, but continuing anyway")
```

Code where error might occur

Error handling code

Without exception handling



With exception handling

More Complex Exception Handling

- We can get details about the specific cause of the exception - i.e. the error message.

```
try:
    x = int("ucd")
except ValueError as e:
    print("Error:",e)
```

```
Error: invalid literal for int()
with base 10: 'ucd'
```

- Try statements can include an optional `finally` clause that always executes regardless of whether an exception occurs.

```
try:
    x = int(some_string)
except ValueError:
    print("Conversion error")
finally:
    print("Always print this")
```

- Multiple except clauses: A try statement can check for several different exception types in sequence.

```
try:
    x = int(some_string)
    answer = x/y
except ValueError:
    print("Conversion error")
except ZeroDivisionError:
    print("Dividing by 0!")
```

Example: Exception Handling

- Common file input tasks required handling the case where we try to read from a file path that does not exist...

```
file_path = "/home/user/data.csv"
try:
    fin = open(file_path,"r")
    content = fin.read()
    print(content)
    fin.close()
except IOError:
    print("Unable to read from file", file_path)
finally:
    print("Process complete")
```

- If `file_path` does not exist, the except code block will be run:

```
Unable to read from file /home/user/data.csv
Process complete
```