Errors & Exception Handling

BIOINF 575

Slides by Ashley Carroll GSI for BIOINF 575 in Fall 2021

Agenda

- **Errors and Exceptions**
- Warnings
- Handling Exceptions
  - try
  - except
  - else
  - finally
- Raising Exceptions

Errors and Exceptions

- Python code stops as soon as it encounters an error

```
File "<ipython-input-4-1d9c06c6d56f>", line 1
  for in [1,2,3]:
      ^
SyntaxError: invalid syntax
```

```
File "<ipython-input-3-9ce6f9fd4ef6>", line 3
  print(i)
  ^
IndentationError: expected an indented block
```

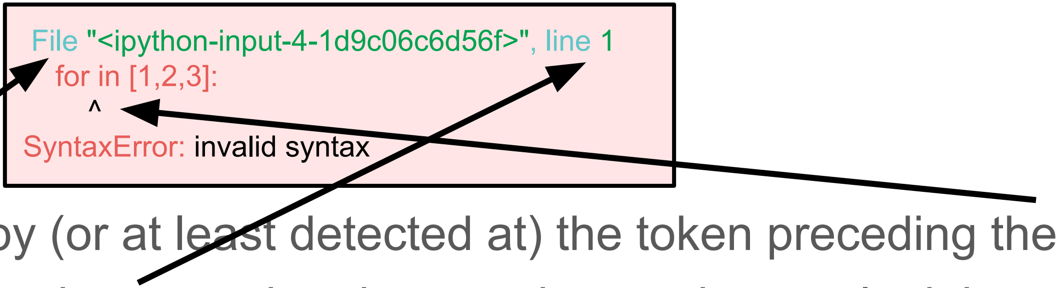
```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-8-e628057d5421> in <module>
      1 d = dict()
----> 2 print(d["A"])

KeyError: 'A'
```

- There are (at least) two distinguishable kinds of errors:
  - syntax errors
  - exceptions

Syntax Errors

- Also known as parsing errors
- The parser repeats the offending line and displays a little ‘arrow’ pointing at the earliest point in the line where the error was detected



```
File "<ipython-input-4-1d9c06c6d56f>", line 1
  for in [1,2,3]:
      ^
SyntaxError: invalid syntax
```

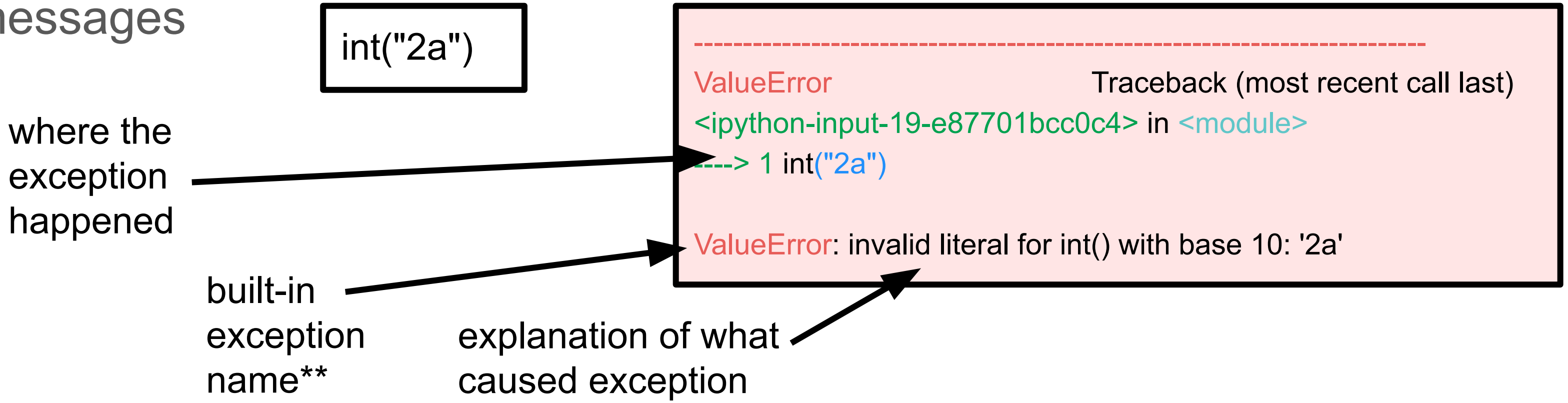
- The error is caused by (or at least detected at) the token preceding the arrow
- File name and line number are printed so you know where to look in case the input came from a script

Exceptions

**disclaimer: all built-in exceptions have string names but user-defined exceptions might not (though it is a useful convention)

- Even if a statement or expression is syntactically correct, it may cause an error when an attempt is made to execute it
- **Errors detected during execution are called exceptions and are not unconditionally fatal (they can be handled)**
- Most exceptions are not handled by programs, however, and result in error messages

```
int("2a")
```



```
---------------------------------------------------------------------------
ValueError                              Traceback (most recent call last)
<ipython-input-19-e87701bcc0c4> in <module>
----> 1 int("2a")

ValueError: invalid literal for int() with base 10: '2a'
```

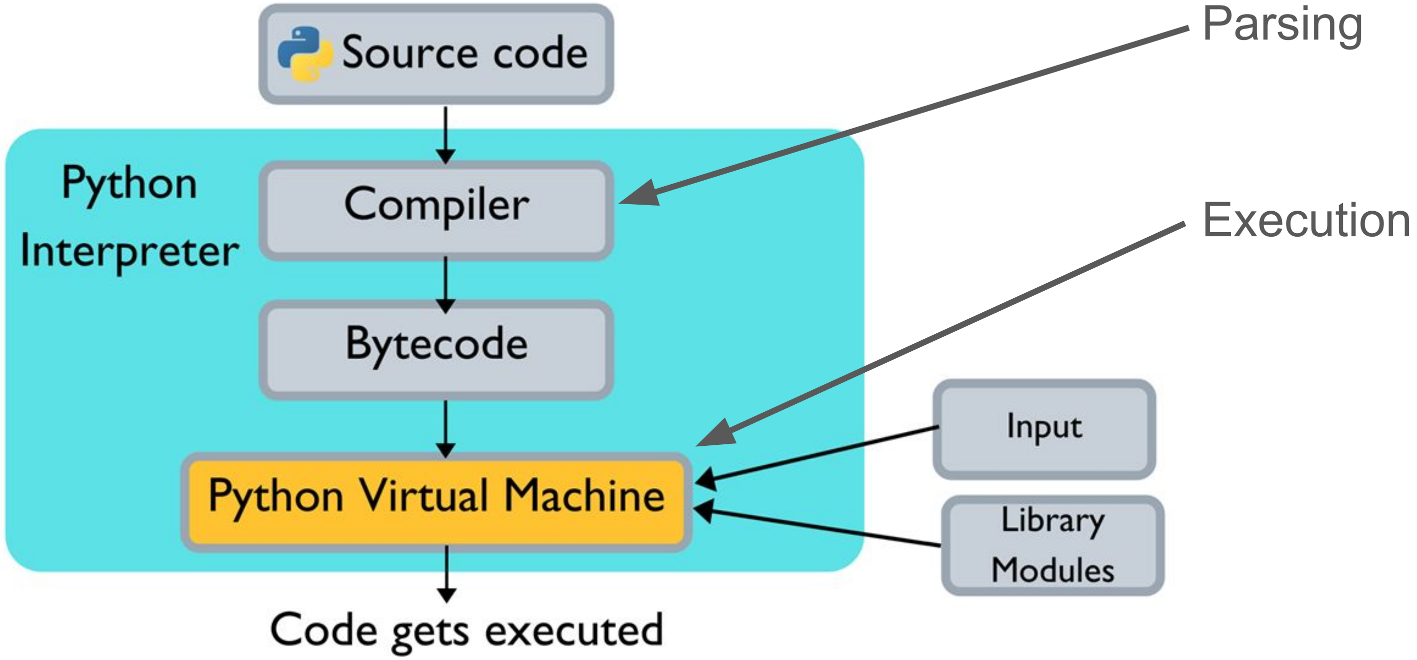
where the exception happened
built-in exception name**
explanation of what caused exception

Errors and Exceptions

Errors	Exceptions
Fatal	Not Unconditionally Fatal
Detected during parsing (syntax errors)	Detected during execution
Cannot be handled	Can be handled

https://docs.python.org/3/tutorial/errors.html

Errors and Exceptions



Exception Hierarchy

https://docs.python.org/3/library/exceptions.html#exception-hierarchy

Python built-in exceptions follow a hierarchical structure:

- An IndentationError IS A SyntaxError
- A ZeroDivisionError IS AN ArithmeticError
    - an ArithmeticError IS AN Exception
    - an Exception IS A BaseException

In Python, all exceptions must be instances of a class that derives from BaseException

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
       |    +-- ZeroDivisionError
     +-- SyntaxError
       |    +-- IndentationError
       |         +-- TabError
```

Agenda

- Errors and Exceptions
- **Warnings**
- Handling Exceptions
  - try
  - except
  - else
  - finally
- Raising Exceptions

Warnings

- Typically issued in situations where it is useful to alert the user of some condition in a program, where that condition (normally) doesn’t warrant raising an exception and terminating the program
    - i.e. one might want to issue a warning when a program uses an obsolete module
- Python programmers issue warnings by calling the warn() function defined in the warnings module
    - https://docs.python.org/3/library/warnings.html
- **Warnings do not stop the code**

Agenda

- Errors and Exceptions
- Warnings
- **Handling Exceptions**
  - **try**
  - **except**
  - else
  - finally
- Raising Exceptions

Handling Exceptions - *try/except*

- Python uses try and except keywords to handle exceptions and both keywords are followed by indented blocks
- If you want to test a block of code for errors -> use the try block
- If you want to handle errors that are raised -> use the except block

General Formula:

```
try:
    # your code here
except AnticipatedError:
    # your code here
```

https://www.pythonforbeginners.com/error-handling/exception-handling-in-python

Handling Exceptions - *try/except*

For the except block, why do I need to specify the type of exception (`AnticipatedError`)?

You technically don't need to...

```
except:
    # your code here
```

This is allowed, but **do not do this**! If you do not specify an exception type, you can catch all exceptions!

Great?

No! Your program will ignore ALL errors and your except block might not be prepared to handle unexpected ones.

Handling Exceptions - *try/except*

Let's revisit the exception hierarchy. If you want to use multiple except statements, you must use the more specific type first.

```
except ZeroDivisionError:
    # your code here
except ArithmeticError:
    # your code here
```

The order here matters! If you reversed the order, the ArithmeticError would catch the ZeroDivisionError (because ZeroDivisionError is a type of ArithmeticError).

```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- SyntaxError
      |    +-- IndentationError
      |         +-- TabError
```

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 1

result = None

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 1

result = None

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 1

result = None

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 1

result = **3.0**

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 1

result = 3.0

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 1

result = 3.0

Output:

Successful Division!

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 1

result = 3.0

Output:

```
Successful Division!
```

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 1

result = 3.0

Output:

```
Successful Division!
```

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 1
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 1

result = 3.0

Output:

```
Successful Division!
3.0
```

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 0

result = None

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 0

result = None

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

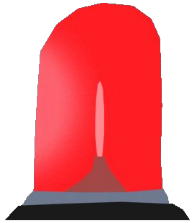
Variables:

numerator = 3

denominator = 0

result = None

Output:



Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 0

result = None

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 0

result = None

Output:

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 0

result = None

Output:

Unsuccessful Division :(

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 0

result = None

Output:

```
Unsuccessful Division :(
```

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

Variables:

numerator = 3

denominator = 0

result = None

Output:

```
Unsuccessful Division :(
```

Handling Exceptions - *try/except*

```
numerator = 3
denominator = 0
result = None

try:
    result = numerator/denominator
    print("Successful Division!")
except ZeroDivisionError:
    print("Unsuccessful Division :(")
    pass

print(result)
```

<u>Variables:</u>

numerator = 3

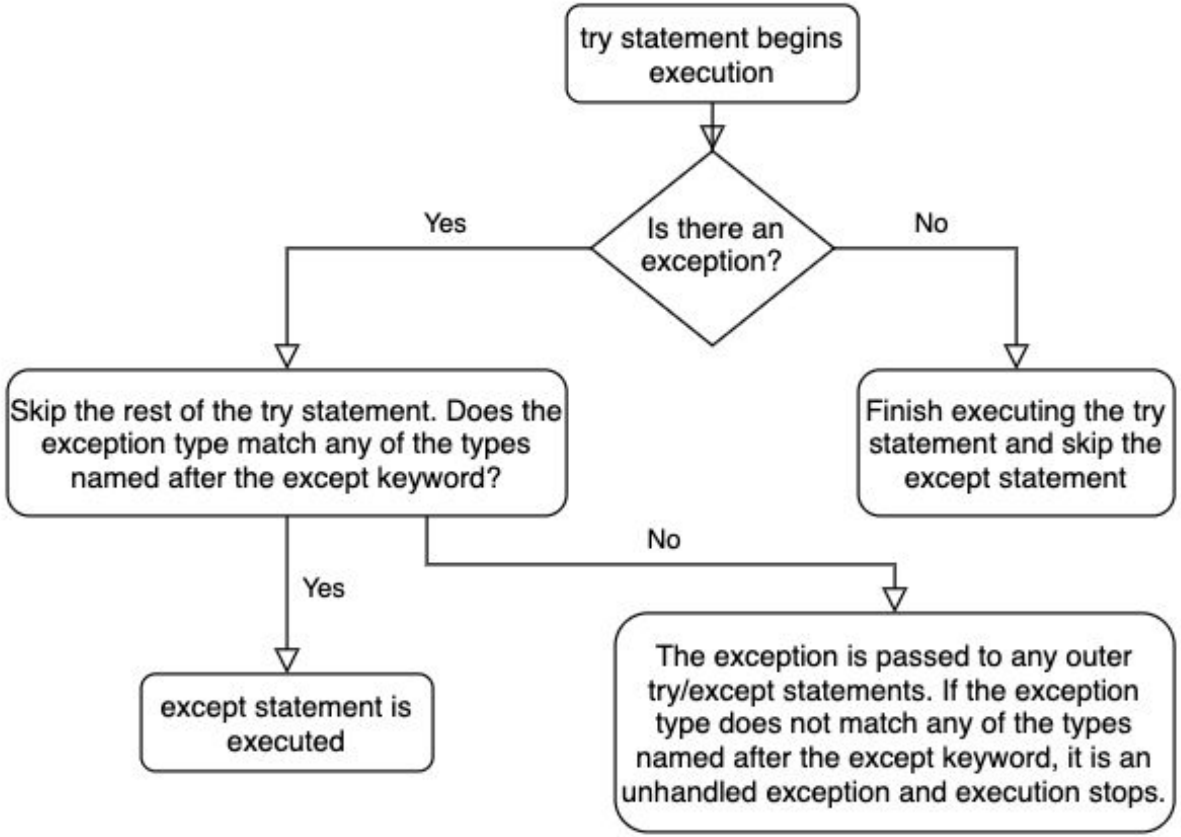
denominator = 0

result = None

<u>Output:</u>

```
Unsuccessful Division :(
None
```

Handling Exceptions - *try/except*


try statement begins execution
Is there an exception?
Yes
No
Skip the rest of the try statement. Does the exception type match any of the types named after the except keyword?
Finish executing the try statement and skip the except statement
Yes
No
except statement is executed
The exception is passed to any outer try/except statements. If the exception type does not match any of the types named after the except keyword, it is an unhandled exception and execution stops.

Agenda

- Errors and Exceptions
- Warnings
- **Handling Exceptions**
  - try
  - except
  - **else**
  - finally
- Raising Exceptions

Handling Exceptions - *else*

- The else statement is an **optional** statement that follows a try/except block
    - it must be after all except statements
- **It will only be executed if the try statement does not raise an exception**

General Formula:

```
try:
    # your code here
except AnticipatedError:
    # your code here
else:
    # your code that will only run if there are no exceptions
```

Agenda

- Errors and Exceptions
- Warnings
- **Handling Exceptions**
    - try
    - except
    - else
    - **finally**
- Raising Exceptions

Handling Exceptions - *finally*

- The finally statement is an **optional** statement that follows a try/except block
- **It will ALWAYS be executed**

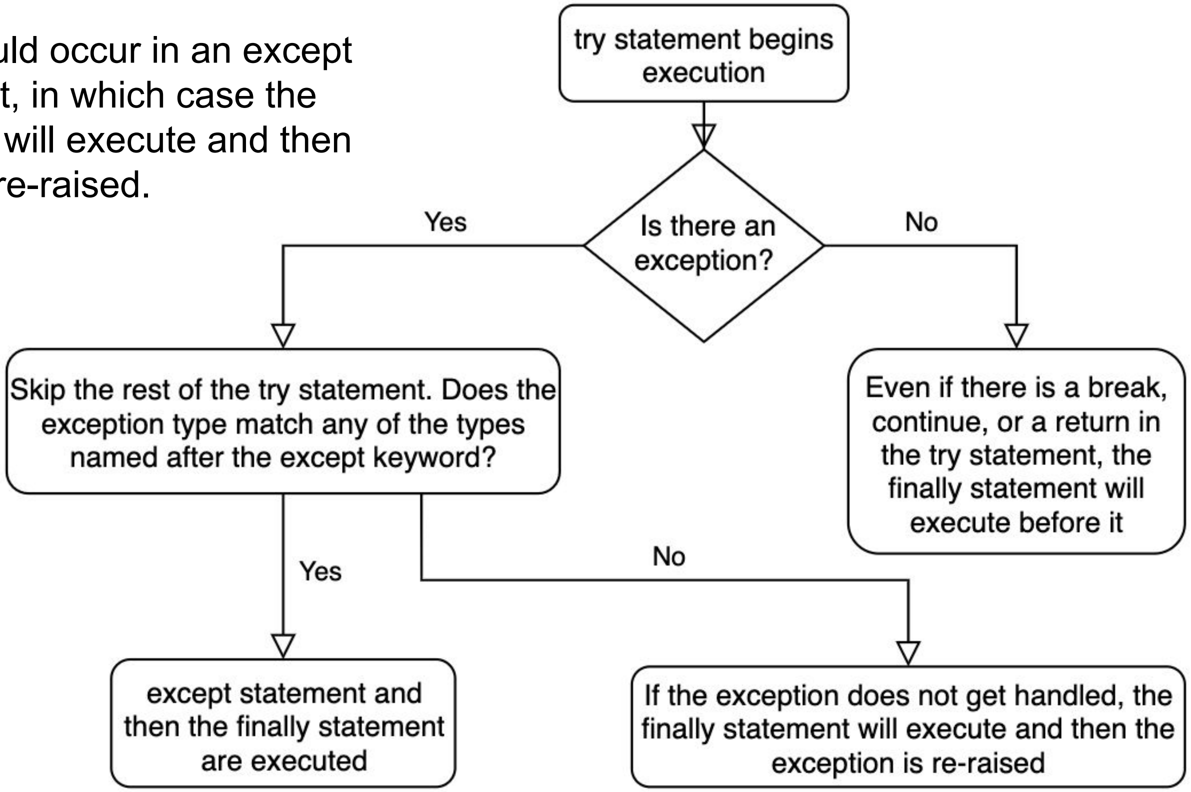
General Formula:

```
try:
    # your code here
except AnticipatedError:
    # your code here
finally:
    # your code that will always run
```

The finally statement can act as a clean-up measure to tie up any loose ends (i.e. close a file that would be left open otherwise)

Handling Exceptions - *finally*

An exception could occur in an except or else statement, in which case the finally statement will execute and then the exception is re-raised.


try statement begins execution
Is there an exception?
Yes
No
Skip the rest of the try statement. Does the exception type match any of the types named after the except keyword?
Even if there is a break, continue, or a return in the try statement, the finally statement will execute before it
Yes
No
except statement and then the finally statement are executed
If the exception does not get handled, the finally statement will execute and then the exception is re-raised

Agenda

- Errors and Exceptions
- Warnings
- Handling Exceptions
    - try
    - except
    - else
    - finally
- **Raising Exceptions**

Raising Exceptions

The `raise` statement allows the programmer to force a specified exception to occur...

`raise` + exception instance/class + optional message

Example:

```
raise NameError("This is where the relevant message should be")
```