PER SCHOLAS

Errors and Exceptions

Overview

- Introduction to Errors and Exceptions
- Built-in Exceptions
- *raise* statement
- *assert* statement
- *try* and *except* statements
- *else* statement
- *finally* statement

Why Errors Exist

- **Problem**
  - So far, we've been presuming that the users will input exactly what we intend them to. We've also been presuming that our code will only ever be used according to our expectation.
  - As you can imagine, these presumptions are unrealistic.

- **Solution**
  - In order to deal with "something going wrong", Python uses ***Errors*** and ***Exceptions***
  - Accounting for these Exceptions is a huge part of programming.

What are Errors and Exceptions?

- There are (at least) two distinguishable kinds of errors: ***syntax errors*** and ***exceptions***.
  - Error caused by not following the proper structure(syntax) of the language is called ***syntax error*** or ***parsing error***.

```
>>> if a < 3
  File "<stdin>", line 1 if a
     < 3
            ^
SyntaxError: invalid syntax
```

(We can notice here that a colon is missing in the if statement.)

  - 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(at runtime) are called ***exceptions***. They occur, for example, when a file we try to open does not exist(*FileNotFoundError*), dividing a number by zero(*ZeroDivisionError*), module we try to import is not found(*ImportError*) etc.

What are Errors and Exceptions?

- Whenever runtime error occur, Python creates an ***exception*** object. If not handled properly(most exceptions are not handled by default), it prints a traceback to that error along with some details about why that error occurred.

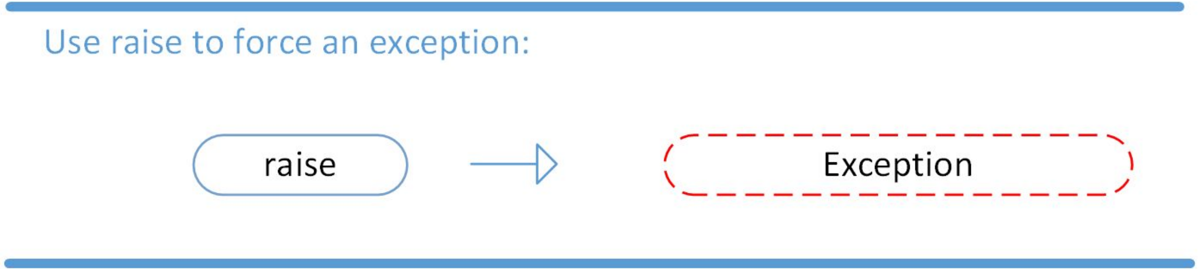
```
>>> 1 / 0
Traceback (most recent call last):
   File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero

>>> open("imaginary.txt")
Traceback (most recent call
last):
   File "<stdin>", line 1, in <module>
FileNotFoundError: [Errno 2] No such file or directory: 'imaginary.txt'
```

Built-in Exceptions

- Python comes with various built-in exceptions, as well as the possibility to create user-defined exceptions. The followings are the few examples:
  - ***ZeroDivisionError*** - Raised when division or modulo by zero takes place for all numeric types
  - ***EOFError*** - Raised when there is no input from input() function and the end of file is reached
  - ***ImportError*** - Raised when an import statement fails
  - ***KeyboardInterrupt*** - Raised when the user interrupts program execution
  - ***IndexError*** - Raised when an index is not found in a sequence
  - ***NameError*** - Raised when an identifier is not found in the local or global namespace
  - ***IOError*** - Raised when an input/output operation fails
  - ***OSError*** - Raised for operating system-related errors
  - ***IndentationError*** - Raised when indentation is not specified properly

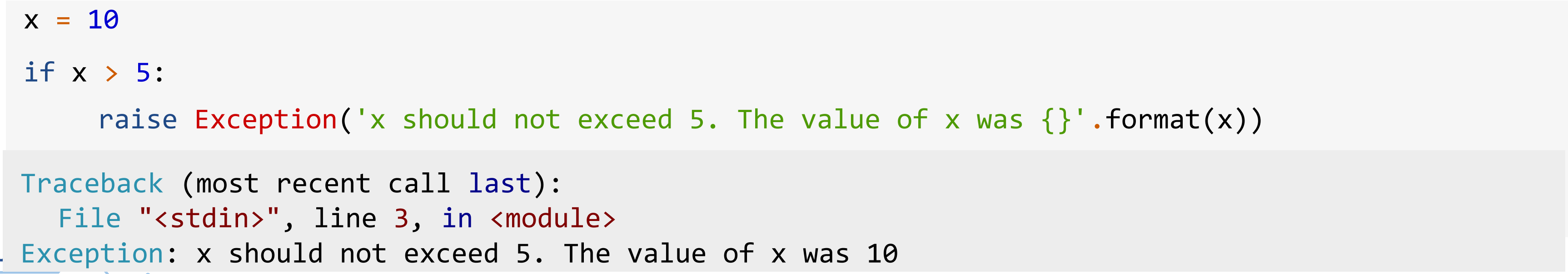
Raising an Exception

- ***raise*** keyword can be used to throw an exception if a condition occurs. The statement can be complemented with a custom exception.



Use raise to force an exception:

- Here's is an example using ***raise*** keyword to throw a custom exception.

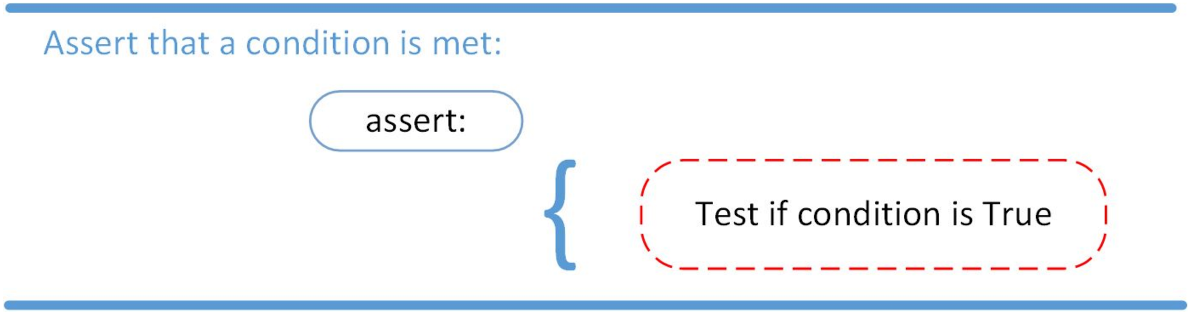


```
x = 10

if x > 5:

     raise Exception('x should not exceed 5. The value of x was {}'.format(x))
```

```
Traceback (most recent call last):
  File "<stdin>", line 3, in <module>
Exception: x should not exceed 5. The value of x was 10
```

The *AssertionError* Exception

- Instead of waiting for a program to crash midway, we can also start by making an assertion in Python. We ***assert*** that a certain condition is met. If this condition is true, then the program can continue. If the condition is false, then the program raises an ***AssertionError*** exception.



Assert that a condition is met:

assert:

Test if condition is True

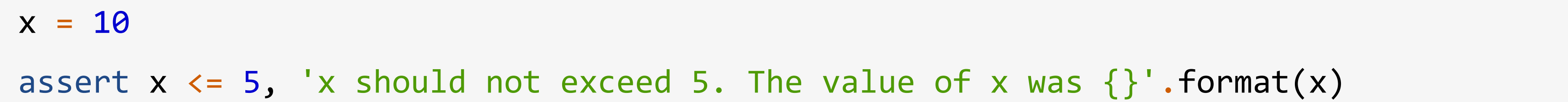
- The syntax for ***assert*** is:

```
assert CONDITION [, ARGUMENT]
```

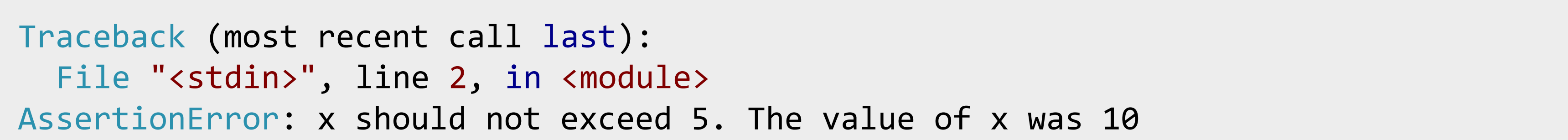
  - If the assertion fails, Python uses *ARGUMENT* as the argument for the AssertionError.

The *AssertionError* Example

- From the example below, the condition is (x <= 5), and the argument is the message. Since the condition is false, the assertion fails and ***AssertionError*** exception is raised with the message.



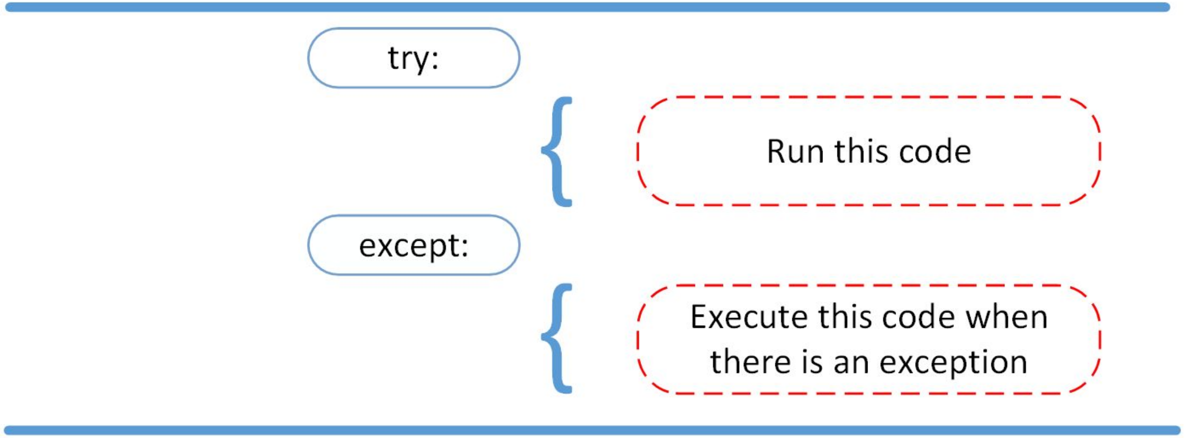
```
x = 10

assert x <= 5, 'x should not exceed 5. The value of x was {}'.format(x)
```



```
Traceback (most recent call last):
  File "<stdin>", line 2, in <module>
AssertionError: x should not exceed 5. The value of x was 10
```

The *Try* and *Except* Block

- The ***try*** and ***except*** block is used to catch and **handle exceptions**. Python executes code following the ***try*** statement and if there are any exception raised from the code, then, Python responds with the code that follows the ***except*** statement. The code in the ***try*** clause will stop executing as soon as an exception is encountered.



try:

Run this code

except:

Execute this code when there is an exception

The *Try* and *Except* Block

- The syntax for ***try*** and ***except*** is:

```
try:
      Code to be executed
except:
      Code to be executed when exceptions raise
```

- When syntactically correct code runs into an error, Python will throw an exception error. This exception error will crash the program if it is unhandled. The ***except*** clause determines how the program responds to exceptions.
- The following function, *modify_x()* only runs if *x* is less than equal to 5

```
def modify_x():
     assert x <= 5, 'x should not exceed 5. The value of x was {}'.format(x) print('x is
     being modified')
```

- Let's give the function ***try***.

The *Try* and *Except* Block Example

```
try:

    x = 10
    modify_x()

except:

    pass
```

□ Since x is greater than 5, *AssertionError* is raised and caught by ***except*** statement and outputs nothing(*pass* statement).

```
try:
    x = 10
    modify_x()
except:
    print('modify_x() function was not executed successfully')
```

□ Since x is greater than 5, *AssertionError* is raised and caught by ***except*** statement and prints *'modify_x() function was not executed successfully'*.

The *Try* and *Except* Block Example

- ❑ In order to see exactly what went wrong, the exact error thrown by the function needs to be caught.

```
try:

     x = 10

     modify_x()

except AssertionError as error:

     print(error)

     print('modify_x() function was not executed successfully')
```

□ Since x is greater than 5, *AssertionError* is raised and caught by ***except*** statement and prints
'x should not exceed 5. The value of x was 10'
'modify_x() function was not executed successfully'.

The *Try* and *Except* Block Example

❑ Multiple ***except*** clauses can be used to handle exceptions after ***try*** clause.

```
try:
     x = 10
     modify_x()
     f = open('my_file.txt')
except FileNotFoundError as fnf_error:
     print(fnf_error)
     print('my_file.txt does not exist') except
AssertionError as error:

     print(error)
     print('modify_x() function was not executed successfully')
```

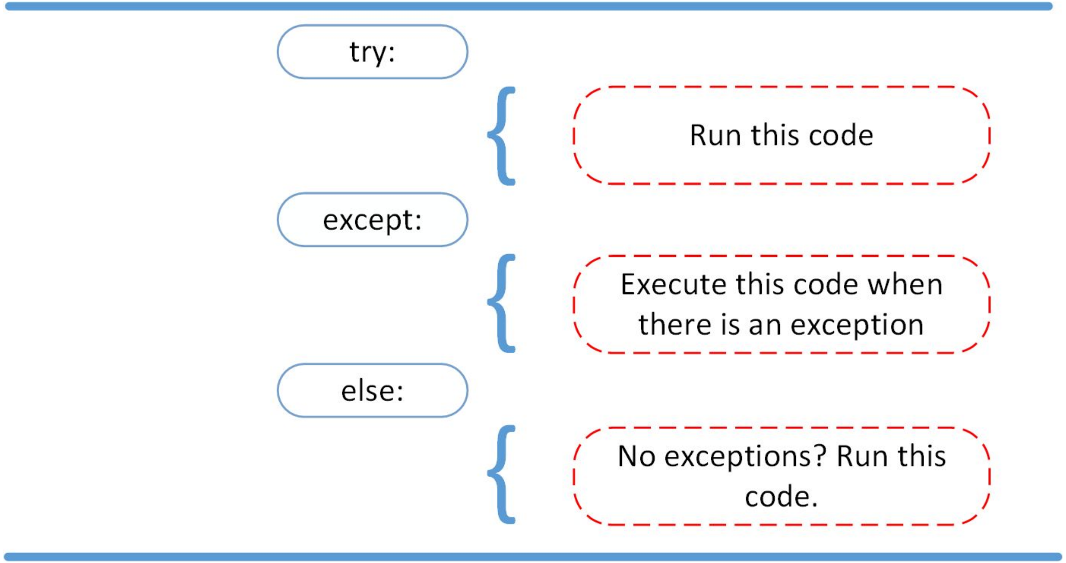
If *x* is greater than 5, only *AssertionError* is raised and caught.

 If *x* is less than or equal to 5 and *my_file.txt* doesn't exist, only *FileNotFoundError* is raised and caught.

 If *x* is less than or equal to 5 and *my_file.txt* exists, the code runs without any exceptions.

The *Else* Clause

- In Python, using the ***else*** statement, the program can be instructed to execute a certain block of code only in the absence of any exceptions.



try:

Run this code

except:

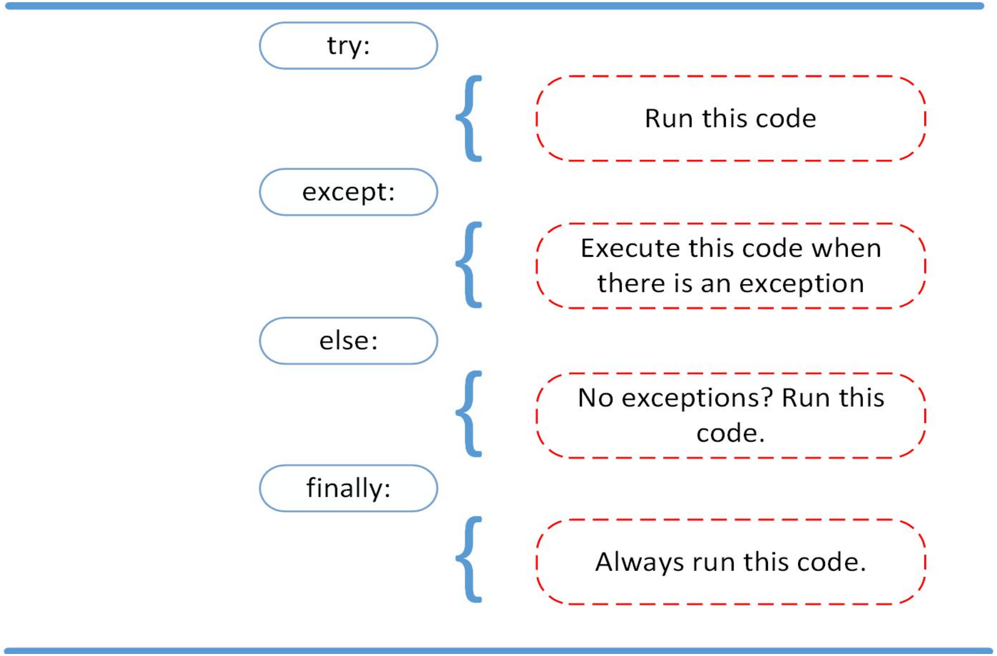
Execute this code when there is an exception

else:

No exceptions? Run this code.

The *Finally* Clause

- When there are some sort of action that need to be implemented to clean up after executing the code, the ***finally*** clause can be used.



try:

Run this code

except:

Execute this code when there is an exception

else:

No exceptions? Run this code.

finally:

Always run this code.

Summary Review

- Introduction to Errors and Exceptions
- Built-in Exceptions
- *raise* allows throwing an exception at any time
- *assert* enables verifying if a certain condition is met and throwing an exception if it isn't
- *try* clause's code is executed until an exception is encountered
- *except* is used to catch and handle the exception(s) that are encountered in the *try* clause
- *else* allows code sections that should run only when no exceptions are encountered in the *try* clause
- *finally* enables executing code sections that should run, with or without any previously encountered exceptions

Questions?