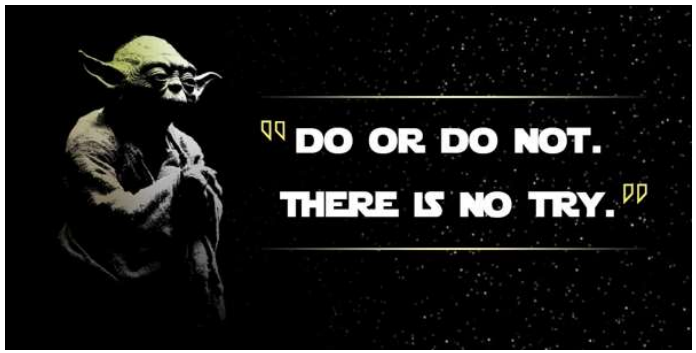
Lecture 07: Exception Handling

Stephen Huang

March 21, 2023



Contents

If you fail to plan (for failure),
you are planning to fail.

Benjamin Franklin

1. Exception

- While converting from `string` to `int`, you may get a `ValueError` exception.
- This exception occurs if the string you want to convert does not represent a number.

```
x = int(input("Please enter a number: "))
```

```
Enter a positive number: n
Traceback (most recent call last):
  File "C:\Users\Stephen\Dropbox\CS\Code\1306\Exception\Exc
    n = int(input("Enter a positive number: "))
ValueError: invalid literal for int() with base 10: 'n'
```

Errors

- A Python program terminates as soon as it encounters an error.
- In Python, an error can be
  - A syntax error, or
  - an exception.
- Syntax errors are the most common complaint while learning a programming language.
- By now, you should be able to "handle" this type of error with ease.

Exception Errors

- Exception Error occurs whenever syntactically correct Python code results in an error.

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

- Two types of exceptions:
  - Built-In Error
  - User-Defined Error (not discussed in this course)

2. Exception Handling

- Errors detected during execution are called exceptions and are not unconditionally fatal. A statement may work for some data and fail for others, hence the name exception.
- We will soon learn how to "handle" them in Python.
- If your program does not handle an exception, it results in an error message, and your program execution stops.

An exception is not fatal; it just has to be handled.

Built-In Exceptions

- There is a list of all built-in exceptions on the python.org website: https://docs.python.org/3/library/exceptions.html#bltin-exceptions.
- Some examples:
  - ZeroDivisionError
  - NameError
  - TypeError
  - ValueError

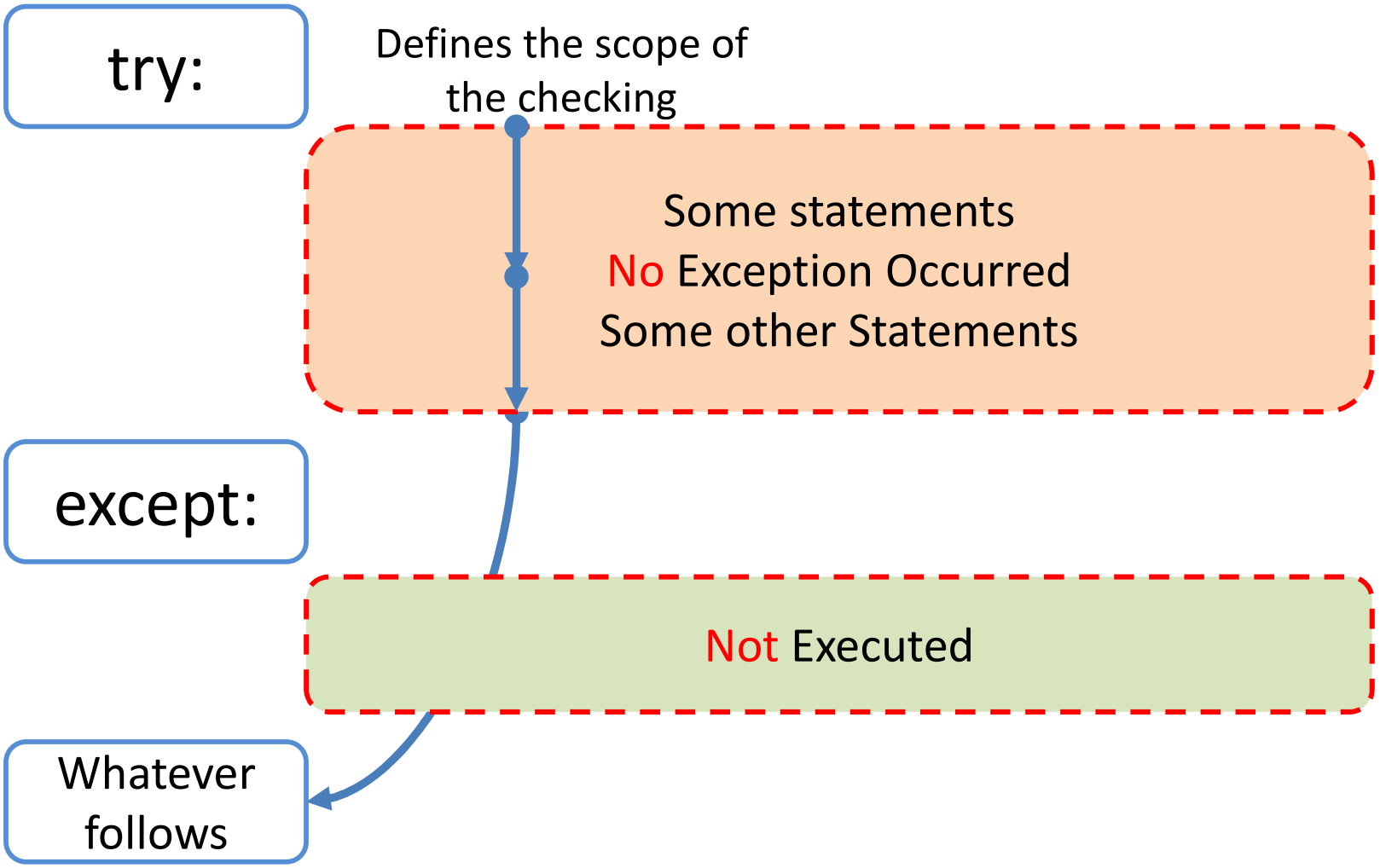
```
BaseException
 +-- SystemExit
 +-- KeyboardInterrupt
 +-- GeneratorExit
 +-- Exception
      +-- StopIteration
      +-- StopAsyncIteration
      +-- ArithmeticError
      |    +-- FloatingPointError
      |    +-- OverflowError
      |    +-- ZeroDivisionError
      +-- AssertionError
      +-- AttributeError
      +-- BufferError
      +-- EOFError
      +-- ImportError
      |    +-- ModuleNotFoundError
      +-- LookupError
      |    +-- IndexError
      |    +-- KeyError
      +-- MemoryError
      +-- NameError
      |    +-- UnboundLocalError
      +-- OSError
      |    +-- BlockingIOError
      |    +-- ChildProcessError
      |    +-- ConnectionError
      |    |    +-- BrokenPipeError
      |    |    +-- ConnectionAbortedError
      |    |    +-- ConnectionRefusedError
```

Handling Exceptions

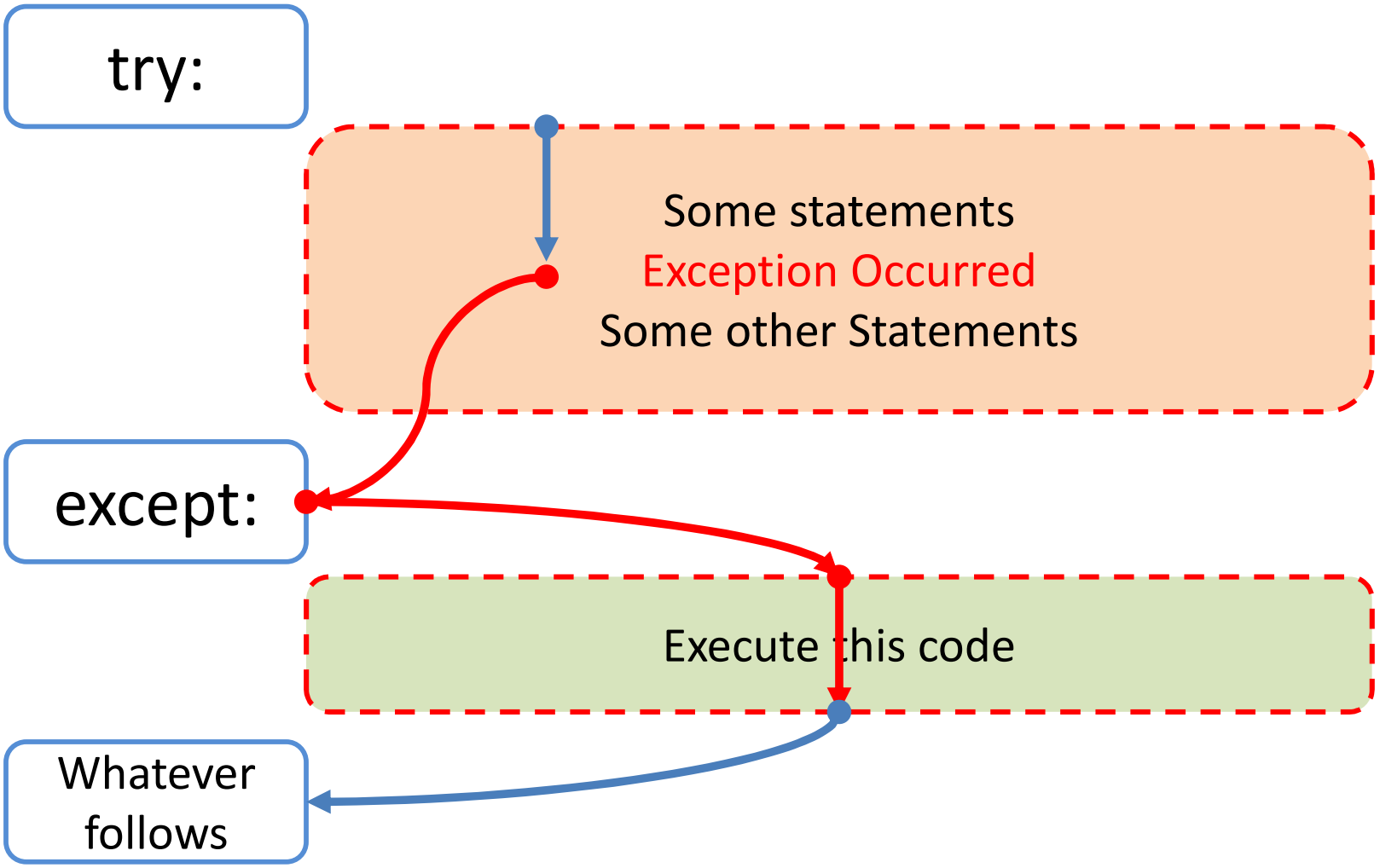
- It is possible to write programs that handle selected exceptions.
- The following example asks the user for input until a valid integer has been entered.

```
while True:
    try:
        x = int(input("Please enter a number: "))
        break
    except ValueError:
        print("Oops!  That was no valid number.  Try again...")
```

Try-Except


try:
Defines the scope of the checking
Some statements
No Exception Occurred
Some other Statements
except:
Not Executed
Whatever follows

Try-Except


try:
Some statements
Exception Occurred
Some other Statements
except:
Execute this code
Whatever follows

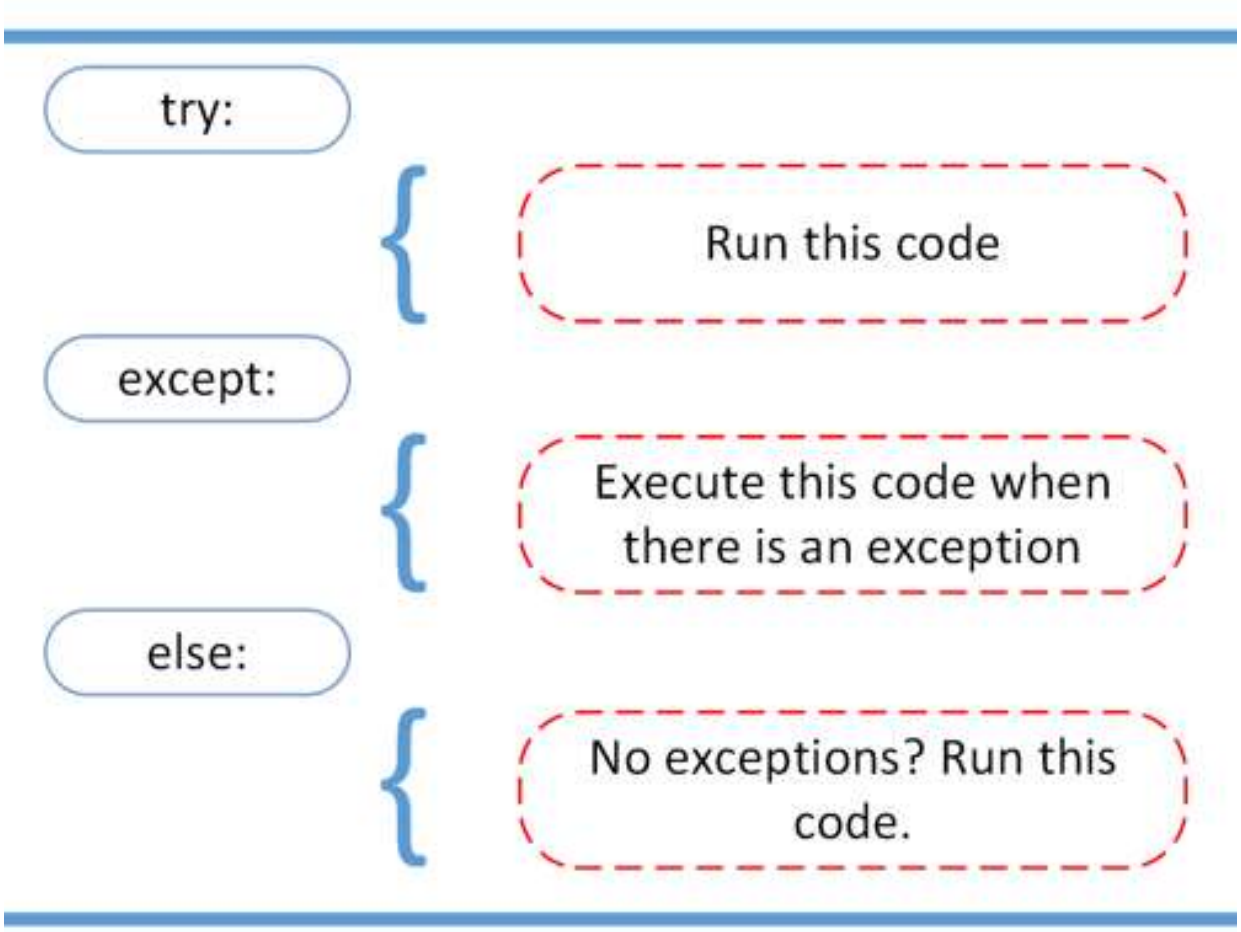
Terminology

- Exceptions
  - Occur
  - Throw
  - Raise

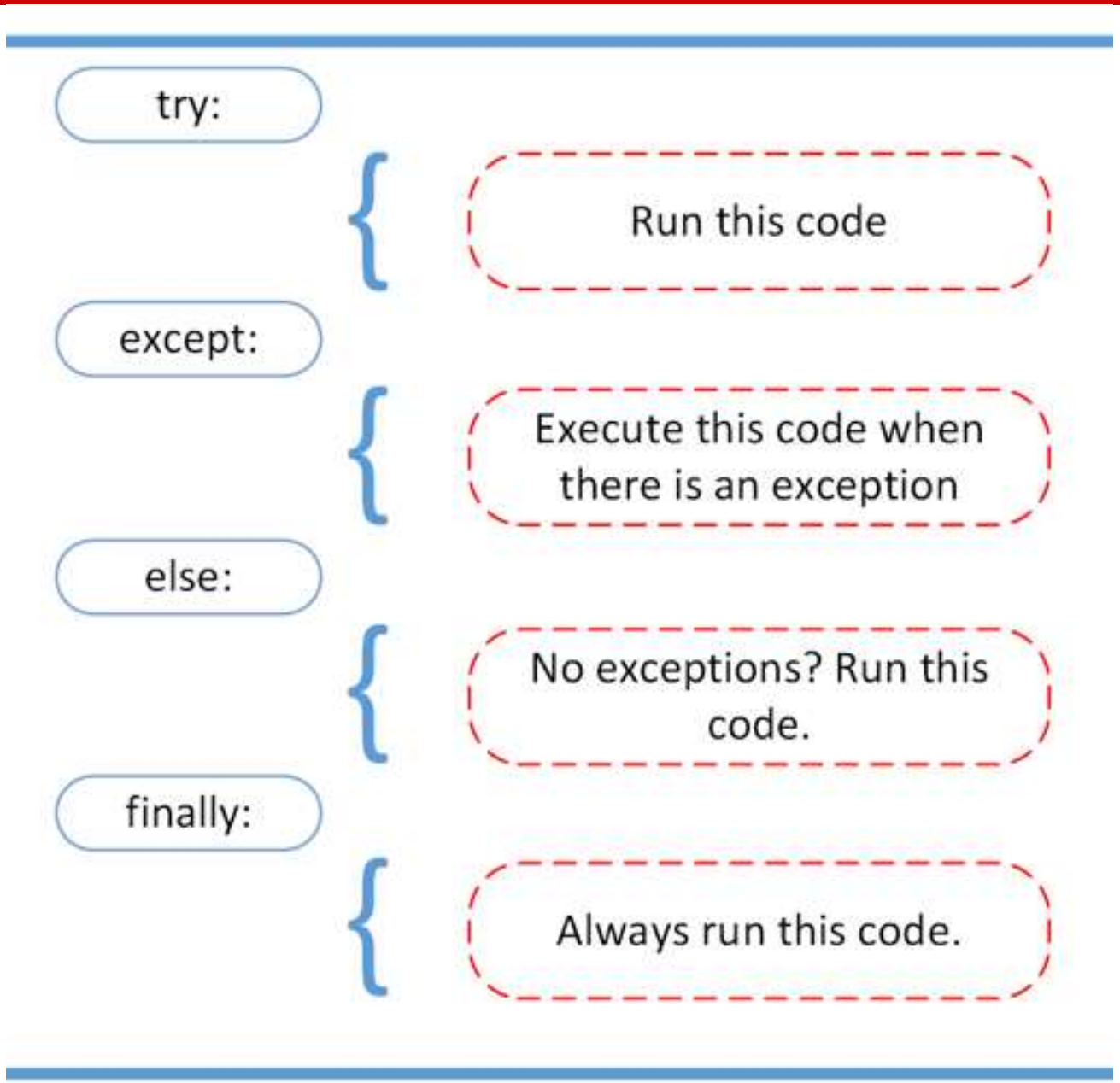
- Exceptions
  - Handle
  - Catch

Try-Except-Else

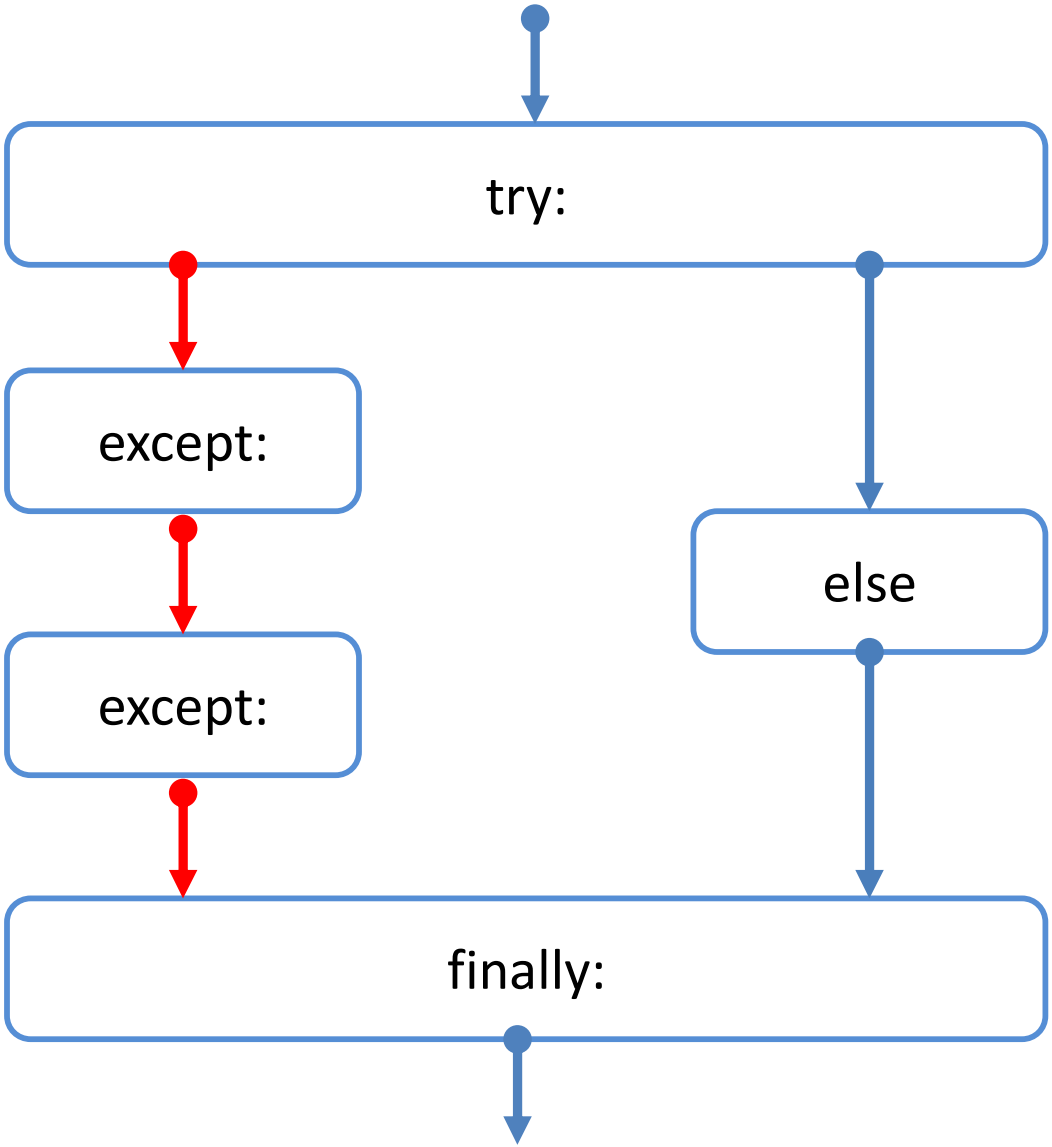
May have multiple excepts for a different type of exception



Try-Except-Else-Finally



Flow



3. Assert and Raise

- Two more commands:
  - Assert (condition): make sure specific condition is met
  - Raise Exception: so that the issue can be handled.
- A defensive programmer will likely put such a statement at the beginning of a function. The purpose is to verify that the parameters are given as specified.

Assertion

- Instead of waiting for a program to crash midway, you can start by making an "assertion" in Python.
- We assert that a specific condition is met.
- If this condition turns out to be True, that is wonderful! The program can continue.
- If the condition turns False, the program can throw an `AssertionError` exception. It's better to get the bad news sooner.

Example

- ```
  assert(0<=x<=100)
  print("Wonderful!")
  ```

- Assert test if the condition is True or not.
  - If True, it continues on the following line.
  - If False, it throws an `AssertionError` exception.
  ```

Assert-Except

```
try:
    x = int(input("Please enter an int: "))
    assert(0<=x<=100)
    print("Wonderful!")
except AssertionError:
    print(f'X = {x} is outside the range.')
```

Raise

- The raise statement allows the code to force a specified exception to occur.
  - `raise <exception>`

```
temp = 101
if temp>=100:
    raise Exception('Too hot')
```

Remarks

- We can give an exception a short alias.
  - `except FileNotFoundError as err:`
- All exceptions are subclasses of `Exception`, and that's what an exception defaults to if not given a specific error.
  - `except:`
- An exception can be re-raised in an exception clause.
  - `raise`

Example

```
try:
    file = open('file.log')
    read_data = file.read()
except FileNotFoundError:
    print(FileNotFoundError)
```