EXCEPTIONS

EXCEPTIONS
HOW TO DEAL WITH PROBLEMS

- Most modern languages provide methods to deal with ‘exceptional’ situations
- It gives the programmer the option to keep the user from having the program stop without warning
  - This is not about fundamental CS, but about doing a better job as a programmer ☺

EXCEPTIONS
WHAT COUNTS AS EXCEPTIONAL

- Errors
  - indexing past the end of a list
  - trying to open a nonexistent file
  - fetching a nonexistent key from a dictionary
  - etc.
- Events
  - search algorithm doesn’t find a value (not really an error)
  - mail message arrives
  - queue event occurs

EXCEPTIONS
ERROR NAMES

Errors have specific names, and Python shows them to us all the time.

```
>>> input_file = open("no_such_file.txt", 'r')
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    input_file = open("no_such_file.txt", 'r')
IOError: [Errno 2] No such file or directory: 'no_such_file.txt'
>>> my_int = int('a string')
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    my_int = int('a string')
ValueError: invalid literal for int() with base 10: 'a string'
>>>
```

EXCEPTIONS ERROR NAMES

- Exceptions keep watch on a particular section of code
  - if an exception occurs, that exception is raised/thrown
  - Then that exception looks for a catcher that can handle that kind of exception
  - If such a handler is found it handles the exception, otherwise Python "handles" it (which usually crashes the program)

EXCEPTIONS

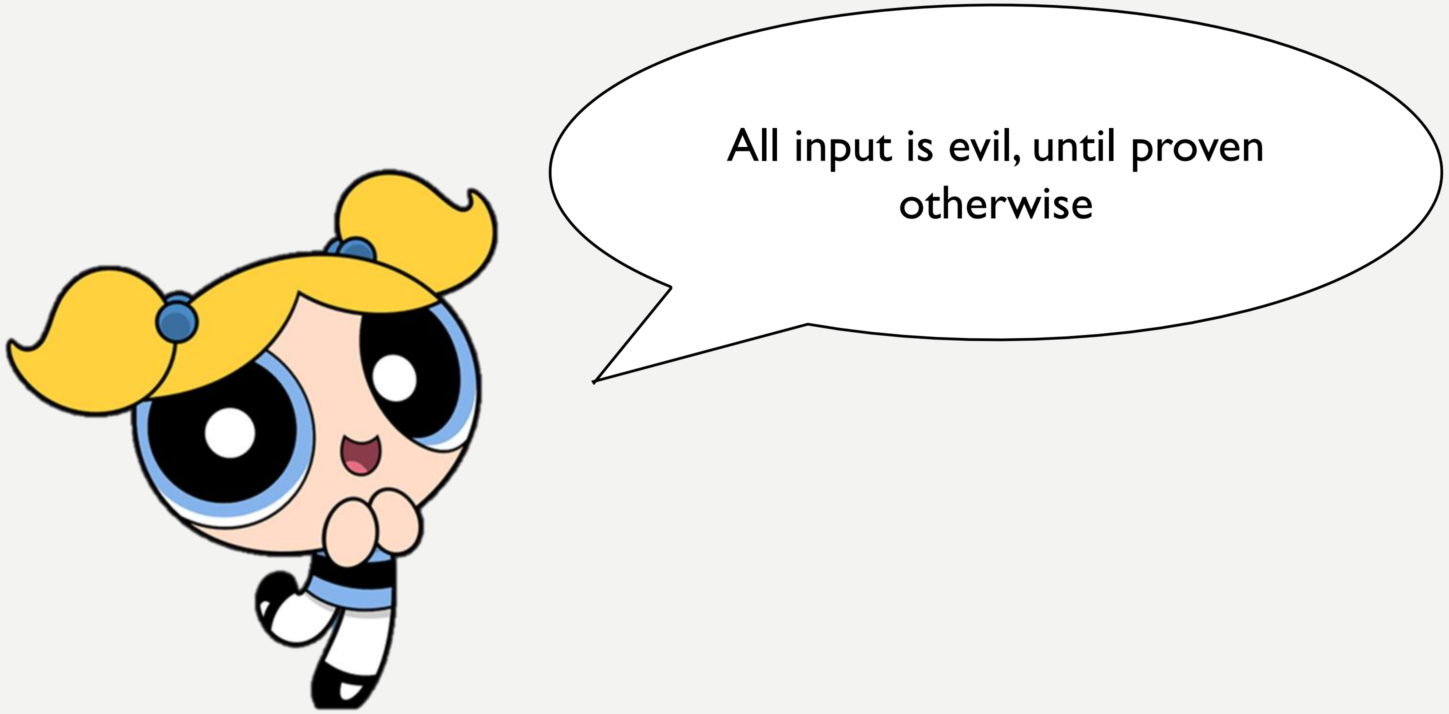
- Doing better with input
  - In general, we have assumed that the input we receive is correct (from a file or from the user)
  - This is almost never true. There is always the chance that the input could be wrong
  - Our programs should be able to handle wrong input

EXCEPTIONS

- Worse yet, input can be evil
  - "Writing Secure Code", by Howard and LeBlanc
    - "All input is evil until proven otherwise"
  - Most security holes in programs are based on assumptions programmers make about input
  - Secure programs protect themselves from evil input

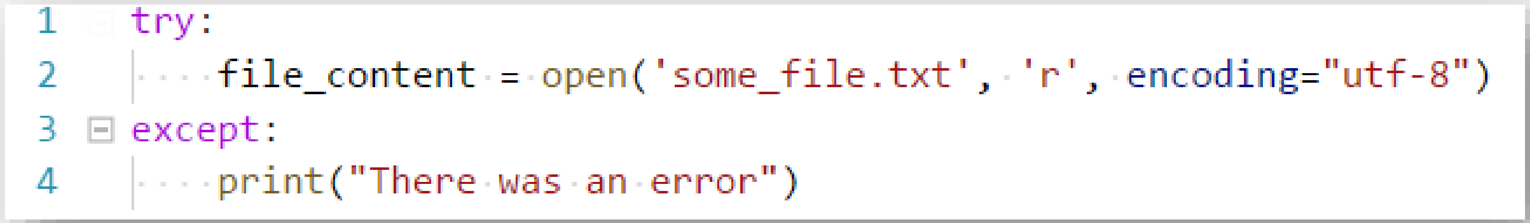
EXCEPTIONS
ERROR NAMES

Rule 7 from the course text book



EXCEPTIONS

```
try:
  suite
except a_particular_error:
  suite
```



```python
try:
    file_content = open('some_file.txt', 'r', encoding="utf-8")
except:
    print("There was an error")
```

EXCEPTIONS

- the `try` suite contains code that we want to monitor for errors during its execution
- if an error occurs anywhere in that `try` suite, Python looks for a handler that can deal with that particular error
  - if no special handler exists, Python handles the error, which usually means that the program crashes with an error message

EXCEPTIONS

- an `except` suite (perhaps multiple `except` suites) is associated with a `try` suite
- each exception names a type of exception it is monitoring for
- if the error that occurs in the `try` suite matches the type of an exception, then that `except` suite is executed

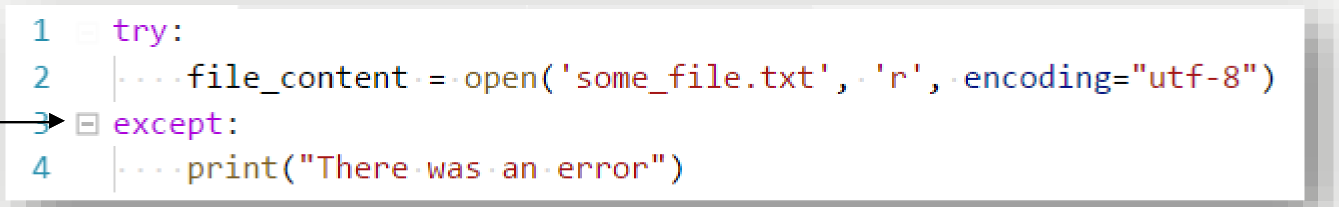
EXCEPTIONS

- if no exception occurs in the `try` suite, all the `try/except` suites are skipped and execution continues on the next line of code after the last exception
- if an error occurs in a `try` suite, Python looks for the right exception
  - if found, that except suite is run
- if no exception handling is found, Python will handle the error

EXCEPTIONS

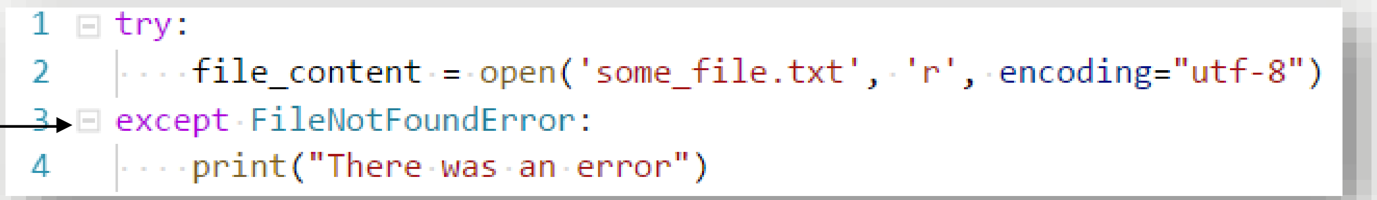
- examples

This except block will catch any error



```python
try:
    file_content = open('some_file.txt', 'r', encoding="utf-8")
except:
    print("There was an error")
```

This except block will only catch FileNotFound errors



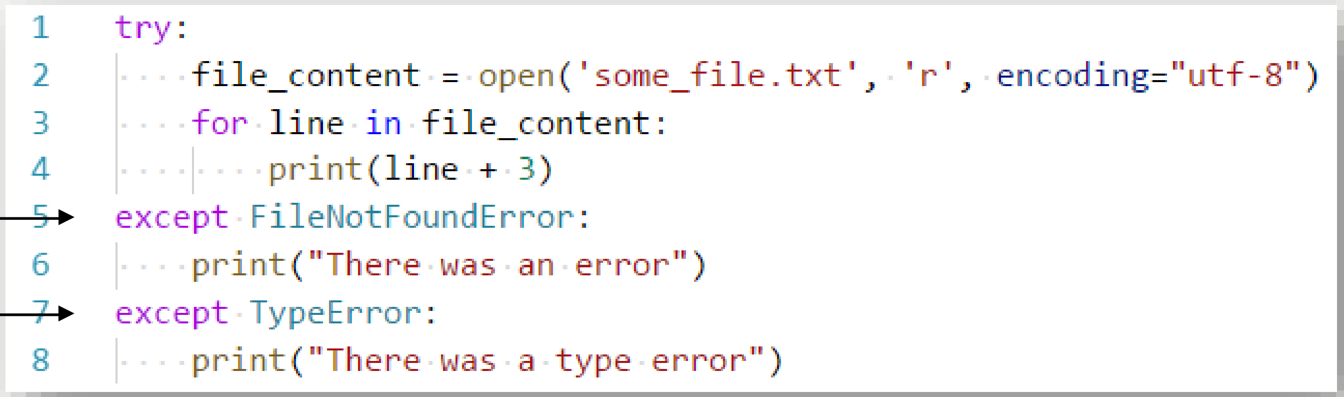
```python
try:
    file_content = open('some_file.txt', 'r', encoding="utf-8")
except FileNotFoundError:
    print("There was an error")
```

EXCEPTIONS

- More examples

This except block will only catch FileNotFound errors

This except block will only catch TypeError errors

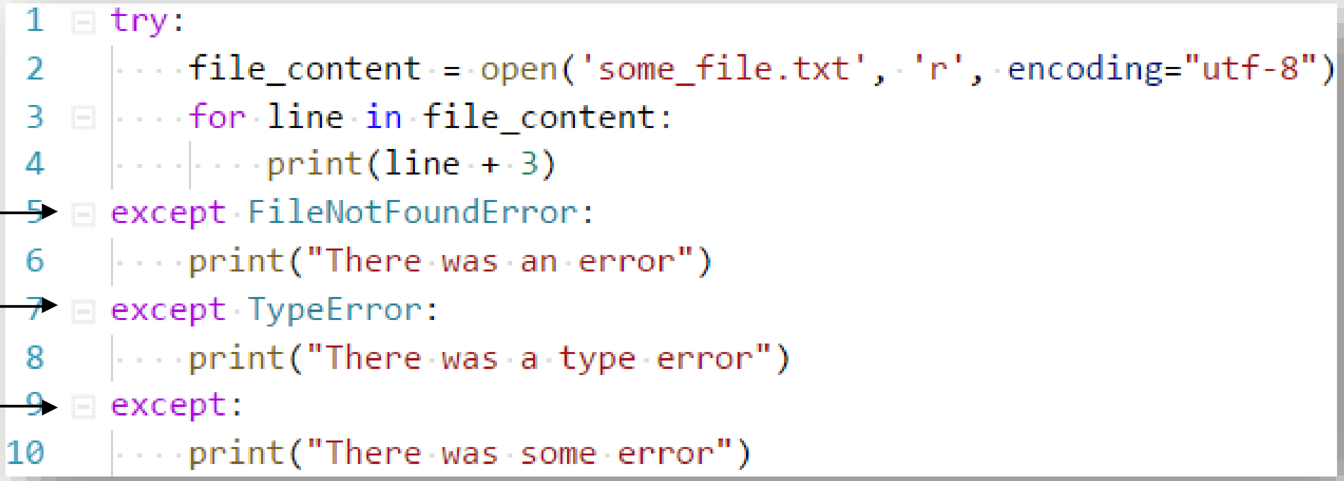


```python
try:
    file_content = open('some_file.txt', 'r', encoding="utf-8")
    for line in file_content:
        print(line + 3)
except FileNotFoundError:
    print("There was an error")
except TypeError:
    print("There was a type error")
```

This except block will only catch FileNotFound errors

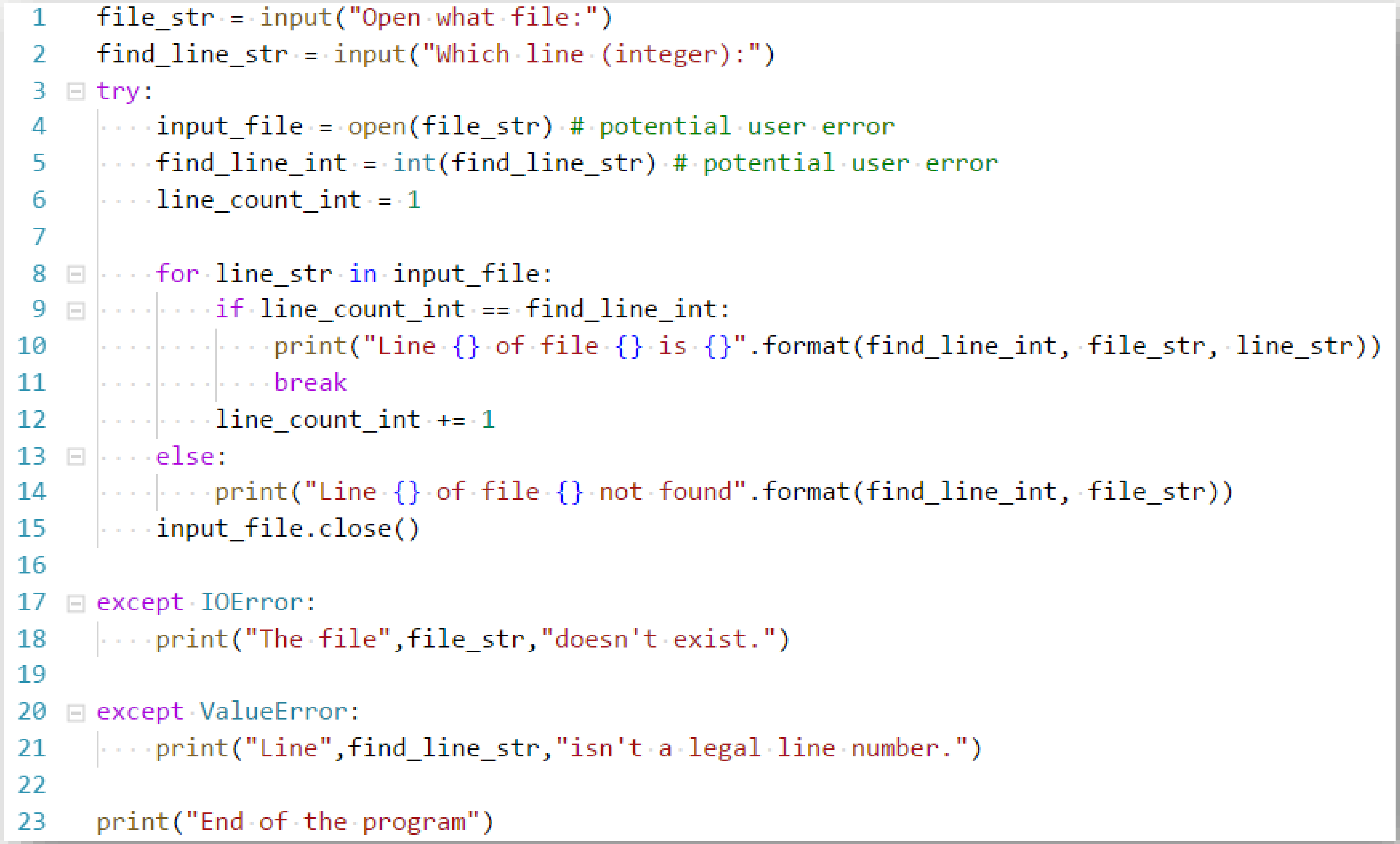
This except block will only catch TypeError errors

This except block will catch all errors



```python
try:
    file_content = open('some_file.txt', 'r', encoding="utf-8")
    for line in file_content:
        print(line + 3)
except FileNotFoundError:
    print("There was an error")
except TypeError:
    print("There was a type error")
except:
    print("There was some error")
```

EXCEPTIONS



```python
file_str = input("Open what file:")
find_line_str = input("Which line (integer):")
try:
    input_file = open(file_str) # potential user error
    find_line_int = int(find_line_str) # potential user error
    line_count_int = 1

    for line_str in input_file:
        if line_count_int == find_line_int:
            print("Line {} of file {} is {}".format(find_line_int, file_str, line_str))
            break
        line_count_int += 1
    else:
        print("Line {} of file {} not found".format(find_line_int, file_str))
    input_file.close()

except IOError:
    print("The file",file_str,"doesn't exist.")

except ValueError:
    print("Line",find_line_str,"isn't a legal line number.")

print("End of the program")
```