Lecture 2.4 : Exception handling

Introduction

- Our programs will encounter errors. When something goes wrong we do not want our programs to simply fall over. We want them to be robust to all circumstances that may arise at runtime. How can our programs cope with runtime errors?
- When something goes wrong at runtime the Python interpreter will *raise an exception*. To be robust to runtime errors our code must accept that they will arise from time to time and *handle* resultant exceptions when they are raised. Here are examples of some runtime errors:

```
# Try to convert 'cat' to an integer
int('cat')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Input In [2], in <module>
      1 # Try to convert 'cat' to an integer
----> 2 int('cat')

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

```
# Try to divide by zero
3/0
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
Input In [3], in <module>
      1 # Try to divide by zero
----> 2 3/0

ZeroDivisionError: division by zero
```

```
# Try to open a file that does not exist
open('no-such-file.txt', 'r')
```

```
---------------------------------------------------------------------------
FileNotFoundError                         Traceback (most recent call last)
Input In [4], in <module>
      1 # Try to open a file that does not exist
----> 2 open('no-such-file.txt', 'r')

FileNotFoundError: [Errno 2] No such file or directory: 'no-such-file.txt'
```

File processing with no exception handling

- Let's look at what happens when a program that processes student results is passed the name of a file that does not exist.

```
#!/usr/bin/env python3

import sys

def main():

    f = open(sys.argv[1], 'r')

    for line in f:
        tokens = line.strip().split()
        mark = int(tokens[-1])
        name = ' '.join(tokens[:-1])

        if mark >= 40:
            result = 'passed'
        else:
            result = 'failed'

        print(f'{name} {result} with a mark of {mark}')

    f.close()

if __name__ == '__main__':
    main()
```

```
$ python3 procfile_v01.py no-such-file.txt
Traceback (most recent call last):
  File "procfile_v01.py", line 24, in <module>
    main()
  File "procfile_v01.py", line 7, in main
    f = open(sys.argv[1], 'r')
FileNotFoundError: [Errno 2] No such file or directory: 'no-such-file.txt'
```

Handling the FileNotFoundError exception

- Rather than have a program abruptly exit on encountering such an error (the default behaviour) Python allows programmers to handle such scenarios gracefully with a `try-except` construct.
- In the `try` block of code we place the instructions that may fail due to a runtime error.
- In the `except` block of code we place the instructions to be carried out in the event of the `try` block failing due to a runtime error.
- If no error arises in the `try` block then execution continues at the instruction following the `try-except` (the contents of the `except` block are ignored).
- If an error occurs within the `try` block, execution stops at that point and the rest of the `try` block is ignored. An exception corresponding to the specific error that has arisen is *raised*. Python then searches for an `except` block that can handle the exception.
- If a suitable `except` block is found it is executed and execution continues from the point following the `try-except`.
- We have updated our program below to handle the file not found error gracefully.

```
#!/usr/bin/env python3

import sys
```

```
def main():

    try:
        f = open(sys.argv[1], 'r')

        for line in f:
            tokens = line.strip().split()
            mark = int(tokens[-1])
            name = ' '.join(tokens[:-1])

            if mark >= 40:
                result = 'passed'
            else:
                result = 'failed'

            print(f'{name} {result} with a mark of {mark}')

        f.close()

    except FileNotFoundError:
        print(f'The file {sys.argv[1]} does not exist.')

if __name__ == '__main__':
    main()
```

```
$ python3 procfile_v02.py no-such-file.txt
The file no-such-file.txt does not exist.
```

What about other exceptions?

- Our code is not yet robust to all runtime errors however. For example, let's see what happens if the file we are processing is incorrectly formatted e.g. due to a typographic error it does not contain an integer mark. If no suitable `except` block is found then the default behaviour applies and program execution is halted.

```
$ cat errors.txt
Mary Connolly 76
John Paul Jones 44
Fred Higgins e0
Laura Timmons 57
Fernandinho 22
```

```
$ python3 procfile_v02.py errors.txt
Mary Connolly passed with a mark of 76
John Paul Jones passed with a mark of 44
Traceback (most recent call last):
  File "procfile_v02.py", line 28, in <module>
    main()
  File "procfile_v02.py", line 12, in main
    mark = int(tokens[-1])
ValueError: invalid literal for int() with base 10: 'e0'
```

- Hmm. We have a couple of problems here. Firstly our program is crashing on encountering a illegal mark. Secondly, because the exception causes our program to exit immediately it does so *without closing the file*. That's bad practice.

- Let's handle the second problem first. Can we fix it so that the file is *always* closed i.e. it is closed when the program runs correctly *and* it is closed in the event of an (unhandled) exception?
- Yes. The use of the `with` statement means the file is always closed cleanly irrespective of whether or not an exception is raised. Let's modify our program to use such a `with` statement:

```
#!/usr/bin/env python3

import sys

def main():

    try:
        with open(sys.argv[1], 'r') as f:

            for line in f:
                tokens = line.strip().split()
                mark = int(tokens[-1])
                name = ' '.join(tokens[:-1])

                if mark >= 40:
                    result = 'passed'
                else:
                    result = 'failed'

                print(f'{name} {result} with a mark of {mark}')

    except FileNotFoundError:
        print(f'The file {sys.argv[1]} does not exist.')

if __name__ == '__main__':
    main()
```

Handling illegal marks

- To gracefully handle the `ValueError` exception caused by the presence of an illegal mark in our input file we need a new `except` block. Where should we place it?
- Well that depends on the kind of behaviour we want. If an error occurs do we want to continue processing the remainder of the file following the error or do we want to give up immediately and ignore the rest of the file?
- If we want to give up immediately and discontinue file processing then we need to place our `except` block *outside* the for loop (so we exit the loop when the exception is handled) and we would do something like this:

```
#!/usr/bin/env python3

import sys

def main():

    try:
        with open(sys.argv[1], 'r') as f:

            for line in f:
                tokens = line.strip().split()
                mark = int(tokens[-1])
                name = ' '.join(tokens[:-1])
```

```
                if mark >= 40:
                    result = 'passed'
                else:
                    result = 'failed'

                print(f'{name} {result} with a mark of {mark}')

    except ValueError:
        print(f'Illegal mark encountered: {tokens[-1]}')

    except FileNotFoundError:
        print(f'The file {sys.argv[1]} does not exist.')

if __name__ == '__main__':
    main()
```

```
$ python3 procfile_v04.py errors.txt
Mary Connolly passed with a mark of 76
John Paul Jones passed with a mark of 44
Illegal mark encountered: e0
```

- If we want to report the illegal mark but process the remainder of the file then we need to place a new `try-except` construct *inside* the for loop.
- Staying inside the for loop on encountering an error means we will go on and process the remainder of the file:

```
#!/usr/bin/env python3

import sys

def main():

    try:
        with open(sys.argv[1], 'r') as f:

            for line in f:

                try:
                    tokens = line.strip().split()
                    mark = int(tokens[-1])
                    name = ' '.join(tokens[:-1])

                    if mark >= 40:
                        result = 'passed'
                    else:
                        result = 'failed'

                    print(f'{name} {result} with a mark of {mark}')

                except ValueError:
                    print(f'Illegal mark encountered: {tokens[-1]}')

    except FileNotFoundError:
        print(f'The file {sys.argv[1]} does not exist.')

if __name__ == '__main__':
    main()
```

```
$ python3 procfile_v05.py errors.txt
Mary Connolly passed with a mark of 76
John Paul Jones passed with a mark of 44
Illegal mark encountered: e0
Laura Timmons passed with a mark of 57
Fernandinho failed with a mark of 22
```

The `else` block

- If execution leaves the `try` block *normally* i.e. **not** as a result of an exception and not as a result of `break`, `continue` or `return` then the `else` block (if present) is executed. The `else` block must be placed after all `except` blocks.

```
#!/usr/bin/env python3

import sys

def main():
    try:
        with open(sys.argv[1], 'r') as f:
            for line in f:
                tokens = line.strip().split()
                mark = int(tokens[-1])
                name = ' '.join(tokens[:-1])

                if mark >= 40:
                    result = 'passed'
                else:
                    result = 'failed'

                print(f'{name} {result} with a mark of {mark}')

    except ValueError:
        print(f'Illegal mark encountered: {tokens[-1]}')

    except FileNotFoundError:
        print(f'The file {sys.argv[1]} does not exist.')

    else:
        print('Reached end of file')

if __name__ == '__main__':
    main()
```

```
$ python3 procfile_v06.py results.txt
Mary Connolly passed with a mark of 76
John Paul Jones passed with a mark of 44
Fred Higgins failed with a mark of 30
Laura Timmons passed with a mark of 57
Fernandinho failed with a mark of 22
Reached end of file
```

```
$ python3 procfile_v06.py errors.txt
Mary Connolly passed with a mark of 76
John Paul Jones passed with a mark of 44
Illegal mark encountered: e0
```

The `finally` block

- A `finally` block is often used in conjunction with `try` and `except` blocks. In the `finally` block we place code that we *always* want executed, irrespective of whether an exception occurs or not. Below we augment our program with a `finally` block that prints a summary of all successfully processed lines before exiting:

```
#!/usr/bin/env python3

import sys

def main():
    lines = 0
    try:
        with open(sys.argv[1], 'r') as f:
            for line in f:

                try:
                    tokens = line.strip().split()
                    mark = int(tokens[-1])
                    name = ' '.join(tokens[:-1])

                    if mark >= 40:
                        result = 'passed'
                    else:
                        result = 'failed'

                    print(f'{name} {result} with a mark of {mark}')
                    lines += 1

                except ValueError:
                    print(f'Illegal mark encountered: {tokens[-1]}')

    except FileNotFoundError:
        print(f'The file {sys.argv[1]} does not exist.')

    else:
        print('Reached end of file')

    finally:
        print(f'Lines processed: {lines}')

if __name__ == '__main__':
    main()
```

```
$ python3 procfile_v07.py errors.txt
Mary Connolly passed with a mark of 76
John Paul Jones passed with a mark of 44
Illegal mark encountered: e0
Laura Timmons passed with a mark of 57
Fernandinho failed with a mark of 22
Reached end of file
Lines processed: 4
```

Writing a file

- Let's complete our program such that it writes its output to a file rather than to the screen. Note how we have enhanced the `with` statement to deal with two files:

```
#!/usr/bin/env python3

import sys

def main():
    lines = 0
    try:
        with open(sys.argv[1], 'r') as fin, open(sys.argv[2], 'w') as fout:

            for line in fin:

                try:
                    tokens = line.strip().split()
                    mark = int(tokens[-1])
                    name = ' '.join(tokens[:-1])

                    if mark >= 40:
                        result = 'passed'
                    else:
                        result = 'failed'

                    fout.write(f'{name} {result} with a mark of {mark}\n')
                    lines += 1

                except ValueError:
                    print(f'Illegal mark encountered: {tokens[-1]}')

    except FileNotFoundError:
        print(f'The file {sys.argv[1]} does not exist.')

    else:
        print('Reached end of file')

    finally:
        print(f'Lines processed: {lines}')

if __name__ == '__main__':
    main()
```

```
$ python3 procfile_v08.py errors.txt processed.txt
Illegal mark encountered: e0
Reached end of file
Lines processed: 4
```

```
$ cat processed.txt
Mary Connolly passed with a mark of 76
John Paul Jones passed with a mark of 44
Laura Timmons passed with a mark of 57
Fernandinho failed with a mark of 22
```