Exceptions

MAD 102

Handling errors

- When the user enters incorrect information – how do you handle it?
  - How do you detect that the information is incorrect and what course of action do you take to ensure that your program continues to run?
- Create **error-checking code** – this introduces code that detects and handles errors while the program is executing

Exception Handling

- Special constructs known as exception-handling, handle exceptional conditions
- For example:

Code:

User inputs ‘dog’ instead of a number

```
num1 = int(input('Please enter a number:'))
num2 = int(input('Please enter a second number:'))

sum = num1 + num2
print(f'The sum of {num1} + {num2} = {sum}')
```

```
Please enter a number:dog
Traceback (most recent call last):
  File "/Users/dtakaki/Library/Mobile Documents/com~apple~Clo
EW/Week 11/ExceptionsClassExamples.py", line 2, in <module>
    num1 = int(input('Please enter a number:'))
           ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
ValueError: invalid literal for int() with base 10: 'dog'
```

Try / Except

- If there is a possibility that your code may produce an exception, place the code in a **try** block
  - The try block is marked with the keyword **try**
- Place the solution for handling the exception, in an **except** block
  - The except block is marked with the keyword **except**

Try / Except

```python
try:
    num1 = int(input('Please enter a number:'))
    num2 = int(input('Please enter a second number:'))

    sum = num1 + num2
    print(f'The sum of {num1} + {num2} = {sum}')

except:
    print('Incorrect information provided - please run the program again')
```

```
Please enter a number:2
Please enter a second number:3
The sum of 2 + 3 = 5
```

Correct Information

```
Please enter a number:dog
Incorrect information provided -_please run the program again
```

Incorrect Information

Try / Except

- If no errors occur in the try block (no exceptions) the code progresses as normal
- If an error occurs (an exception is thrown), the code in the exception block is executed.
- Any code in the try block is skipped – **regardless of where the error might have occurred**

Exception Errors

- Value Error – raised when an operation or function receives an argument that has the right type but an inappropriate value

```
print(int('dog'))
```

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

```
letters = ['a','b','c']
let1, let2 = letters
```

```
ValueError: too many values to unpack (expected 2)
```

Exception Errors

- KeyError - Raised when a mapping (dictionary) key is not found in the set of existing keys.

```
teams = {'Red Wings' : 1926, 'Blackhawks': 1926 }
print(teams['Maple Leafs'])
```

```
KeyError: 'Maple Leafs'
```

Exception Errors

- IndexError - Raised when a sequence subscript is out of range.

```
winning_numbers = [1, 2 ,3 ,4]
print(winning_numbers[4])
```

```
IndexError: list index out of range
```

Exception Errors

NameError - Raised when a sequence subscript is out of range.

```
def display_message(name):
    print(f'Hello {name}')

print(name)
```

```
NameError: name 'name' is not defined
```

Zero Division Error

- ZeroDivisionError - Raised when the second argument of a division or modulo operation is zero

```
grade = int(input('Please enter your mark:'))
test_total = int(input('Please enter what the test is out of:'))

average = grade / test_total * 100

print(f'For this test you receive {average:.2f}%')
```

```
Please enter your mark:1
Please enter what the test is out of:0
Traceback (most recent call last):
  File "/Users/dtakaki/Library/Mobile Documents/com~apple~CloudDocs/Developer/MAD 102 - N
EW/Week 11/ExceptionsClassExamples.py", line 52, in <module>
    average = grade / test_total * 100
              ~~~~~~^~~~~~~~~~~~
ZeroDivisionError: division by zero
```

Exception Errors

- More built-in exception errors can be found in the Python documentation
- https://docs.python.org/3/library/exceptions.html

Multiple Exception Handlers

- Depending on your program, multiple exceptions could be generated
- Multiple exception handlers can be added to a try block to handled the various different types of exceptions that may occur
- Each is placed in its own except block and the type of exception it is meant to handle is listed

Multiple Exception Handlers

```
try:
    grade = int(input('Please enter your mark:'))
    test_total = int(input('Please enter what the test is out of:'))

    average = grade / test_total * 100

    print(f'For this test you receive {average:.2f}%')
except ZeroDivisionError:
    print('Invalid test total entered - cannot be 0')
except ValueError:
    print('Invalid entry - please enter a number for the grade and test_total')
```

To handle a 0 being entered for a test total

To handle a non-number being entered in either input

Multiple Exception Handlers

- What happens if another type of error occurs?
- This would result in an unhandled exception
  - Add a generic except block to deal with any additional issues that may arise

Multiple Exception Handlers

- DRY principle – Don't Repeat Yourself
- In situations where different types result in the same exception code being used, exceptions can be grouped using a tuple

```
except (ValueError, TypeError):
    # Code to execute
```

Raising Exceptions

- Exception handling constructs can be used to raise (throw) errors
  - This means identifying that an exception has occurred
- The keyword raise is used, followed by the type of error to to raise
  - A string argument is provided to the exception error that details the issue

Raising Exceptions

- The exception block uses the keyword as to bind the string value that was passed to a new variable

```
except ValueError as reason:
    print(reason)
```

Raising Exceptions

- If a different error of that type occurs, the default messaging is used

```
try:
    grade = int(input('Please enter your mark:'))
    test_total = int(input('Please enter what the test is out of:'))
    if test_total <=0:
        raise ValueError('Test total must be greater than 0')
    average = grade / test_total * 100

    print(f'For this test you receive {average:.2f}%')
except ValueError as reason:
    print(reason)
    print('Incorrect values entered')
```

Only occurs if the test_total value is set to 0 or a value less than 0

Exceptions with functions

- Exceptions can be part of functions – allows for better program clarity

```
def get_testTotal():
    test_total = int(input('Please enter what the test is out of:'))
    if test_total <=0:
        raise ValueError('Test total must be greater than 0')
    return test_total

try:
    grade = int(input('Please enter your mark:'))
    test_total = get_testTotal()
    average = grade / test_total * 100

    print(f'For this test you receive {average:.2f}%')
except ValueError as reason:
    print(reason)
    print('Incorrect values entered')
```

finally clause

- The finally clause of a try statement allows a programmer to specify clean-up actions
  - These actions are always executed
  - This clause always comes last – after the try and except blocks

finally clause

- If no exceptions occur, execution continues in the finally clause and proceeds with the program
- If a handled exception occurs, the exception handler is executed and then the finally clause
- If an unhandled exception occurs, the finally clause is executed, and the exception is reraised
- The finally clause will also execute if any break, continue or return statement causes the try block to be exited

finally clause

```python
try:
    num = int(input('Please enter a number:'))
    incrementor = 5

    result = num + incrementor

    print(f'{num} has increased by {incrementor} to {result}')
except ValueError:
    print('Please enter a value number to increment')
except:
    print('An error has occured, please try again')
finally:
    print('This always happens')
```

Valid values

```
Please enter a number:2
2 has increased by 5 to 7
This always happens
```

Invalid values

```
Please enter a number:d
Please enter a value number to increment
This always happens
```

Creating Custom Error Types

- You can create your own custom exception types
- They are a class
- Create with an initializer and use as normal

Custom Exception Type

```
class IncorrectMovieError(Exception):
    def __init__(self, value):
        self.value = value

try:
    fav_movie = input('Please enter your favourite movie:').title()

    if fav_movie != 'Star Wars':
        raise IncorrectMovieError('You did not enter Star Wars as your favourite movie!')
    else:
        print(f'Your favourite movie is {fav_movie}')
except IncorrectMovieError as reason:
    print(reason)
```

Correct

```
Please enter your favourite movie:Star Wars
Your favourite movie is Star Wars
```

Incorrect

```
Please enter your favourite movie:Star Trek
You did not enter Star Wars as your favourite movie!
```