10-exceptions

September 16, 2022

1 Exceptions

- When an error is detected during execution, it is called an exception
- To handle an exception, Python gives you the `try` statement (as many other languages).
- Entering the `try` block will make Python watch out for one or more different types of exceptions
- Exceptions as raised and we should react to them
- The 'reaction' code is placed in `except` blocks
- The `else` clause (optional), is executed when the try clause is exited without any exception raised
- The `finally` clause (optional) code is executed regardless of whatever happened in the other clauses.

[1]:
```
def try_syntax(numerator, denominator):
    try:
        print(f'In the try block: {numerator}/{denominator}')
        result = numerator / denominator
    except ZeroDivisionError as zde:
        print("---> ZeroDivisionError")
        print(zde)
    except Exception as e:
        # All built-in, non-system-exiting exceptions are derived from this␣
 ↪class.
        # All user-defined exceptions should also be derived from this class.
        # if you simply want to "catch" the exception and get along with it,␣
 ↪you could simple use `except:`
        print("---> Exception")
        print(e)
    else:
        print('The result is:', result)
        return result
    finally:
        print('Exiting')
```

[2]:
```
try_syntax(12, 4)
```

```
In the try block: 12/4
The result is: 3.0
```

```
Exiting
```

```
[2]: 3.0
```

```
[3]: try_syntax(11, 0)
```

```
In the try block: 11/0
---> ZeroDivisionError
division by zero
Exiting
```

```
[4]: try_syntax(11, None)
```

```
In the try block: 11/None
---> Exception
unsupported operand type(s) for /: 'int' and 'NoneType'
Exiting
```

```
[ ]:
```