#16_Errors

29/05/2025

- Many situations when Python cannot exeute program and an **Exception object** is created that describes the problem

- **Exception and Error** are used interchangeably.

NameError

- When attempting to use a variable or function that has not been defined, a **NameError** arises.

```
a  # Referencing an undefined variable results in a NameError
a()  # The same applies when trying to call an undefined function
```

TypeError

- A **TypeError** occurs when a value of the wrong type is used in an expression. For example:

  -using an argument of the wrong type as a function argument

```
>>> my_str = "abc"
>>> my_str.find(42)  # TypeError: must be str, not int
```

  - trying to call an object that is not callable

```
>>> my_int = 42
>>> my_int()      # TypeError: 'int' object is not callable
```

SyntaxError

- A **SyntaxError** occurs when code is encountered by Python that does not meet syntactic rules.

```
print('hello)
# SyntaxError: unterminated string
```

29/05/2025

- SyntaxError typically arises immediately after loading a Python program, before it starts running.

- Unlike NameError and TypeError, which hinge on specific variables and values encountered during the program runtime, Python detects SyntaxErrors solely from the program's text.

```
def (          #SyntaxError: unexpected EOF while parsing
```

- Example of SyntaxError occurring whilst the program is running

```
expression = "2 * (3 + 4"
result = eval(expression)
```

NB: eval function not commonly used.

The eval function evaluates an expression that has been written as a string. No different to

```
result = 2 * (3 + 4)
```

The eval function raises a SyntaxError when code is executed

SyntaxError raised during the parsing phase.

29/05/2025.

ValueError

- A **ValueError** is raised when a function receives an argument of a correct data type, but the value of the argument is inappropriate for the operation

```
number = int("abc")
#ValueError: invalid literal for int() with base 10: 'abc'
```

IndexError

- An **IndexError** occurs when trying to access an index of a sequence (like a list or a string) that is outside of valid indexes.

```
nums = [1,2]
num = nums[2] #IndexError: list index out of range
```

KeyError

- A **KeyError** is raised when trying to access a dictionary key that does not exist.

```
my_dict = {"a": 1, "b": 2}
value = my_dict["c"] #KeyError: 'c'
```

ZeroDivisionError

- A **ZeroDivisionError** occurs when attempting to divide by zero (0, 0.0) or when trying to use 0 on the right side of the modulo (%) operator.

```
result1 = 10/0 #ZeroDivisionError: division by zero
result2 = 42%0 #ZeroDivisionError: integer modulo by zero
```

29/05/2025

Exception Handling

- Exception handling allows you to manage errors that might occur during program execution. These exceptions can be handled using the following statements.
  - try
  - except
  - else
  - finally

  } statements.

Try Block:

- The code that might raise an exception is placed within the try block
- Python will monitor this block for any exceptions that may occur during its execution

Except Block:

- If an exception is raised in the try block, Python will look for a matching except block that can handle that specific type of exception
- If a match is found, the code within the corresponding except block is executed.

Else Block (Optional):

- else block is executed only if no exceptions occurred in the try block
- used for code that should run if no errors are encountered

Finally Block (Optional):

- always executed regardless of whether an exception was raised
- used for cleanup operations or tasks that must be performed, such as releasing resources

29/05/2025

```
try:
    num_str = input("Enter a number: ")
    num = int(num_str)

    result = 10/num
except ValueError:
    print("Invalid input, you didn't enter a number.")
except ZeroDivisionError:
    print("Cannot divide by zero.")
else:
    print(f"Result: {result}")
finally:
    print("Exception handling complete.")
```

- If user enters a non-integer value, a `ValueError` is caught, and the program displays an error message.
- If the user enters zero, a `ZeroDivisionError` is caught, and the program warns about division by zero.
- If no exceptions occur, the division result is printed.
- Regardless of whether an exception was caught or not, the `finally` block ensures that the final message is displayed.