Errors

Table of Contents

1

Introduction to Errors

Introduction (review)

- Consider the following example :

```
1  print("Don't say 'I never make a mistake'"
```

What is the output? Try to figure out in your mind...

Introduction (review)

- Consider the following example :

```
1  print("Don't say 'I never make a mistake'"
```

```
1  Traceback (most recent call last):
2  File "code.py", line 1
3      print("Don't say 'I never make a mistake'"
4                                                ^
5  SyntaxError: unexpected EOF while parsing
```

Introduction (review)

```
1  Traceback (most recent call last):
2  File "code.py", line 1
3      print("Don't say 'I never make a mistake'"
4                                                ^
5  SyntaxError: unexpected EOF while parsing
```

⚠ Attention:

- Do not panic when you see those error lines. Do not hesitate to read carefully what they are saying to you.

2

Syntax Errors

Syntax Errors (review)

```
1  Traceback (most recent call last):
2  File "code.py", line 1
3      print("Don't say 'I never make a mistake'"
4                                                ^
5  SyntaxError: unexpected EOF while parsing
```

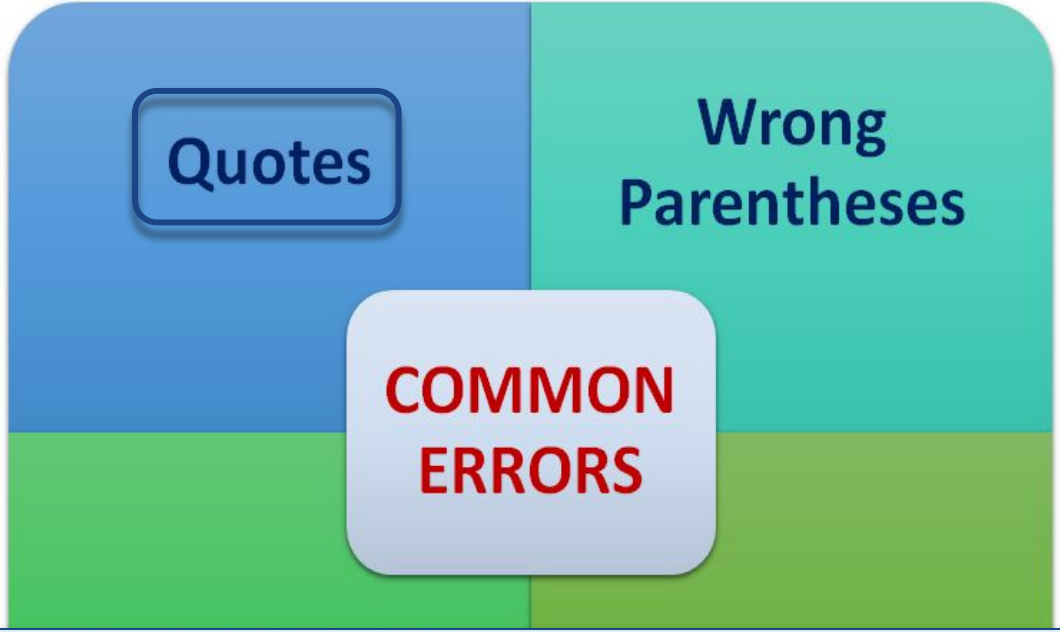
) is forgotten

associated value

3 Common Errors

Common Errors (review)

Common Errors (review)



Tips:

- Keep this simple advice in your mind; triple quotes for multi-line strings, double or single quotes for ordinary strings.

Common Errors (review)

Common Errors (review)

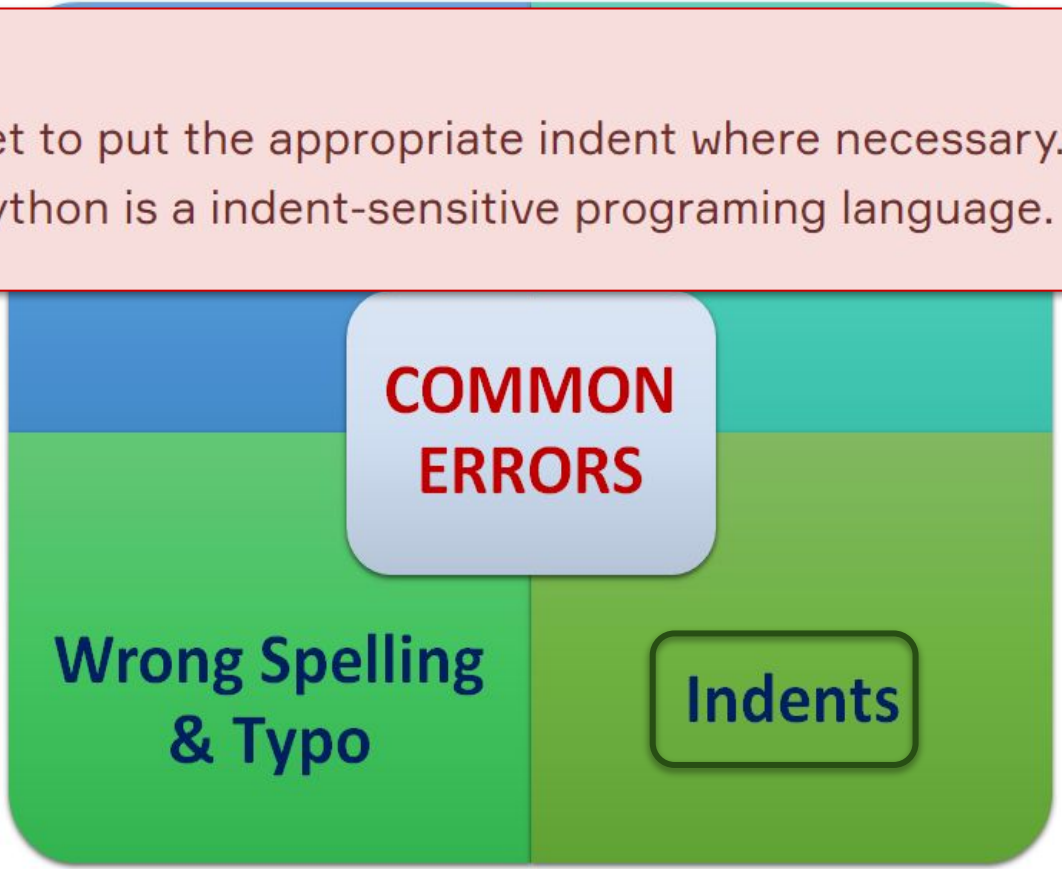
⚠ **Avoid ! :**

- Do not confuse uppercase and lowercase letter of the keywords. Keep in your mind that Python is a case-sensitive programing language.

Common Errors (review)

⚠ **Avoid ! :**

- Do not forget to put the appropriate indent where necessary. Keep in your mind that Python is a indent-sensitive programing language.



Common Errors

- Let's find some errors in the codes :

```
1  status = []
2  if status:
3      print("'''Hello World")
4  else
5      print("Hello Universe'")
6
```

What is the error? Try to figure out in your mind...

Common Errors

- There is a typo (missing colon) :

```
1  status = []
2  if status:
3      print("'Hello World")
4  else
5      print("Hello Universe'")
6
```

a colon **:** should be put here

Output

```
  File "code.py", line 4
    else
       ^
SyntaxError: invalid syntax
```

Common Errors

- Let's find some errors in the codes :

```
1  x = ["1", "2", "3"]
2  y = ["USA", "Japan", "Spain"]
3
4  for i in y:
5      for j in X:
6          print(type([tuple(i+j)]))
7
```

What is the error? Try to figure out in your mind...

Common Errors

- Remember, Python is a case-sensitive language :

```
1  x = ["1", "2", "3"]
2  y = ["USA", "Japan", "
3
4  for i in y:
5      for j in X:
6          print(type([tuple(i+j)]))
7
```

case of "x" should be the same

Output

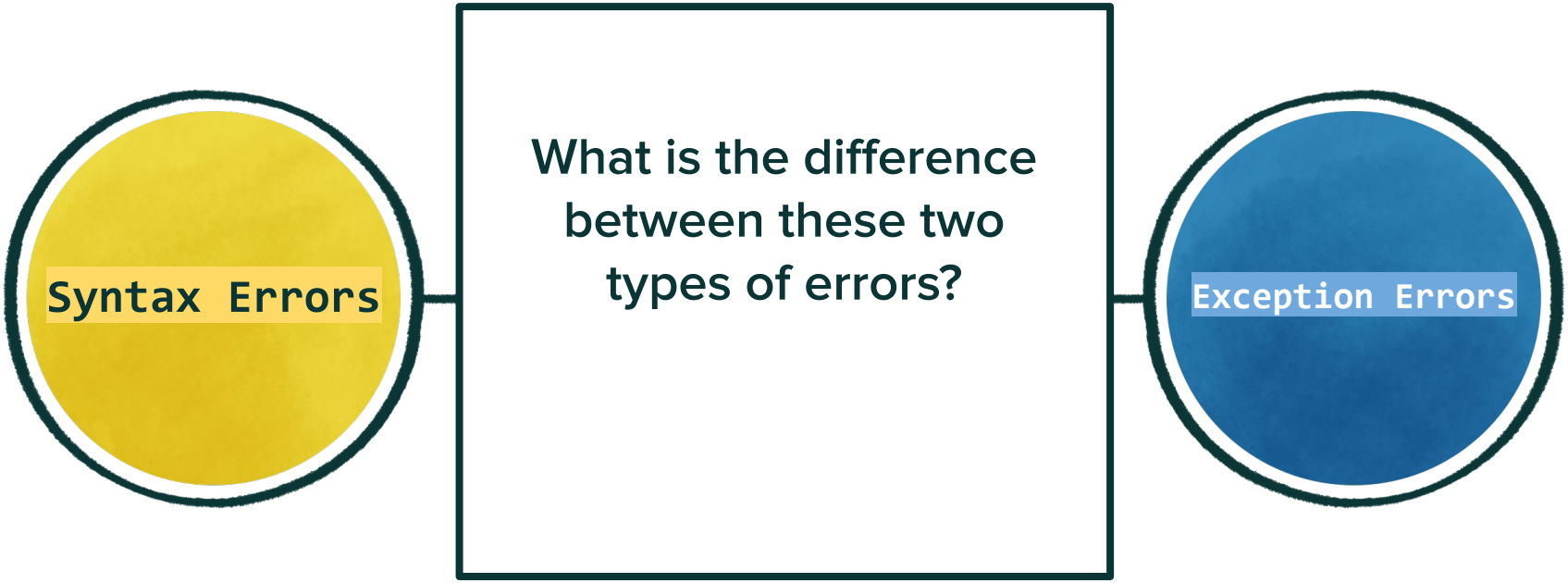
```
Traceback (most recent call last):
  File "code.py", line 5, in <module>
    for j in X:
NameError: name 'X' is not defined
```

Exceptions

Table of Contents

1 Introduction

What is the difference?



Introduction (review)

Syntax Error	Exception Error
These types of errors are detected during **compiling** the program into byte-code.	These types of errors are detected during the program execution (**interpretation**) process.

Introduction (review)

- Now, let's examine the following example :

```
1  print('Here we go!')
2  print('I will be the second text')
3  a = '3'
4  b = 5
5  print('It is time for an error message :(')
6  print(a + b)  # it won't be printed
7  print("Sorry, but I won't be printed")  # it won't ve printed
```

Introduction (review)

- Now, let's examine the following example :

```
1  print('Here we go!')
2  print('I will be the second text')
3  a = '3'
4  b = 5
5  print('It is time for an error message :(')
6  print(a + b)  # it won't be printed
7  print("Sorry, but I won't be printed")  # it won't ve printed
```

```
1  Here we go!
2  I will be the second text
3  It is time for an error message :(
4  Traceback (most recent call last):
5    File "code.py", line 6, in <module>
6      print(a + b)
7  TypeError: can only concatenate str (not "int") to str
```

Introduction (review)

- Exceptions also have explanatory "**associated value**" at the last line of the error message.

```
1  Here we go!
2  I will be the second text
3  It is time for an error message :(
4  Traceback (most recent call last):
5    File "code.py", line 6, in <module>
6      print(a + b)
7  TypeError: can only concatenate str (not "int") to str
```

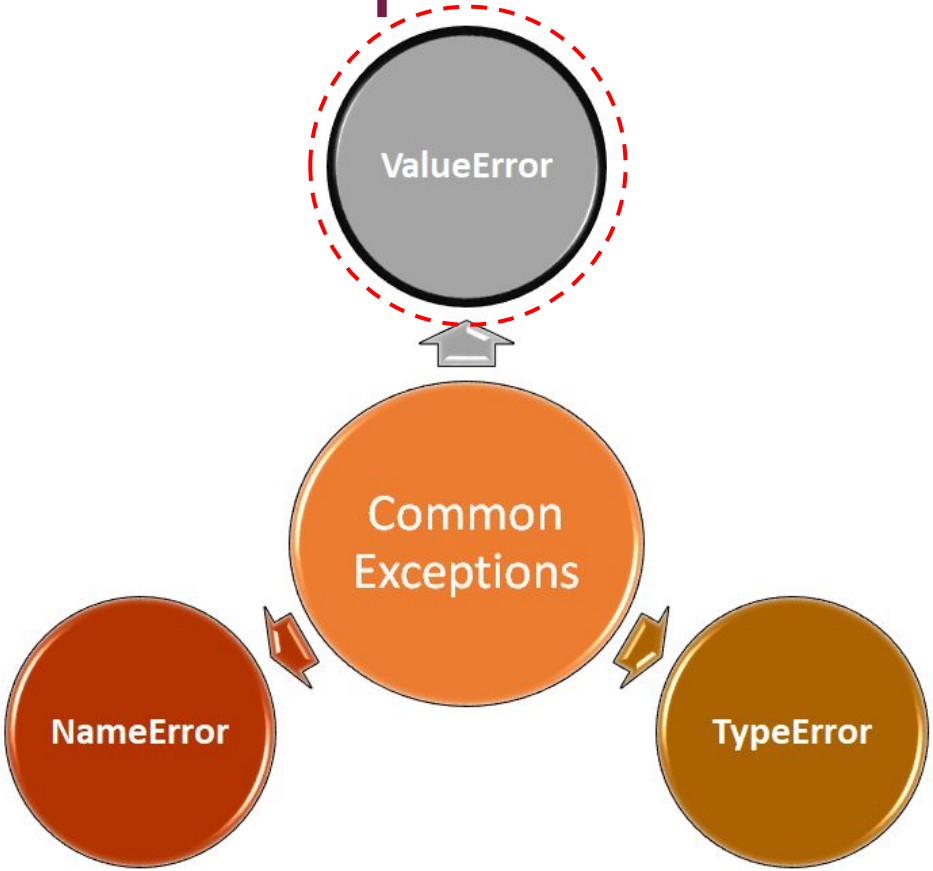
Explanatory text value of the error message. It's known as **"associated value"**

2

Common Exceptions

Common Exceptions (review)

```
ValueError
```

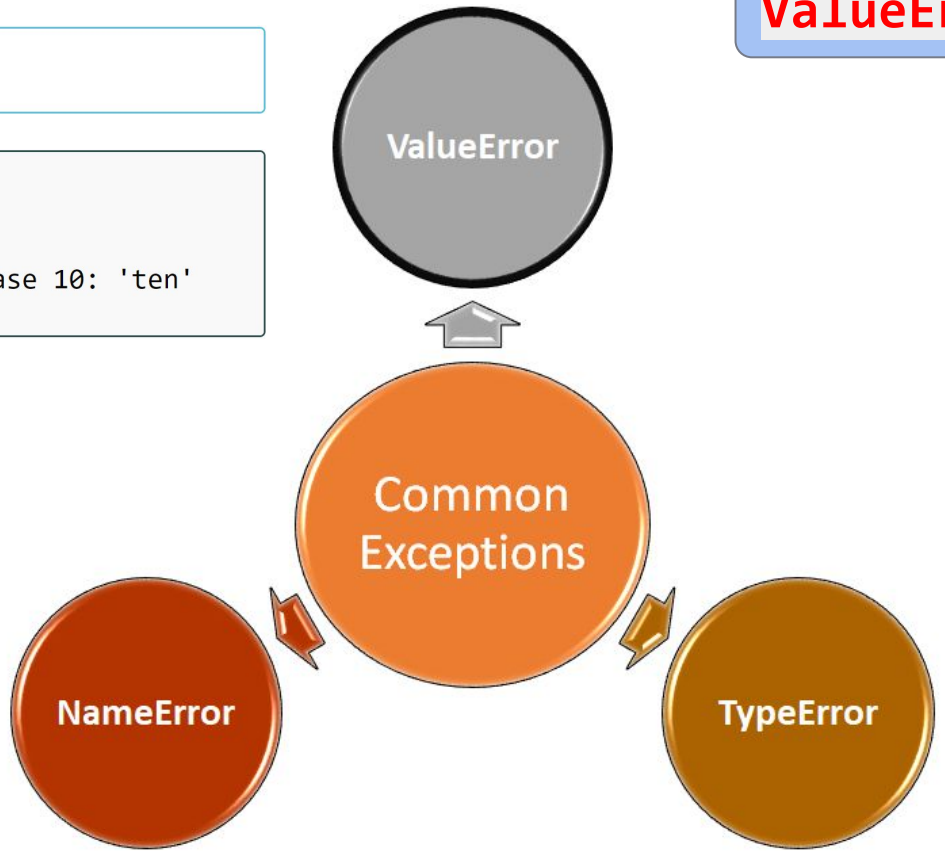


Common Exceptions (review)

```
1  print(int('ten'))
2
```

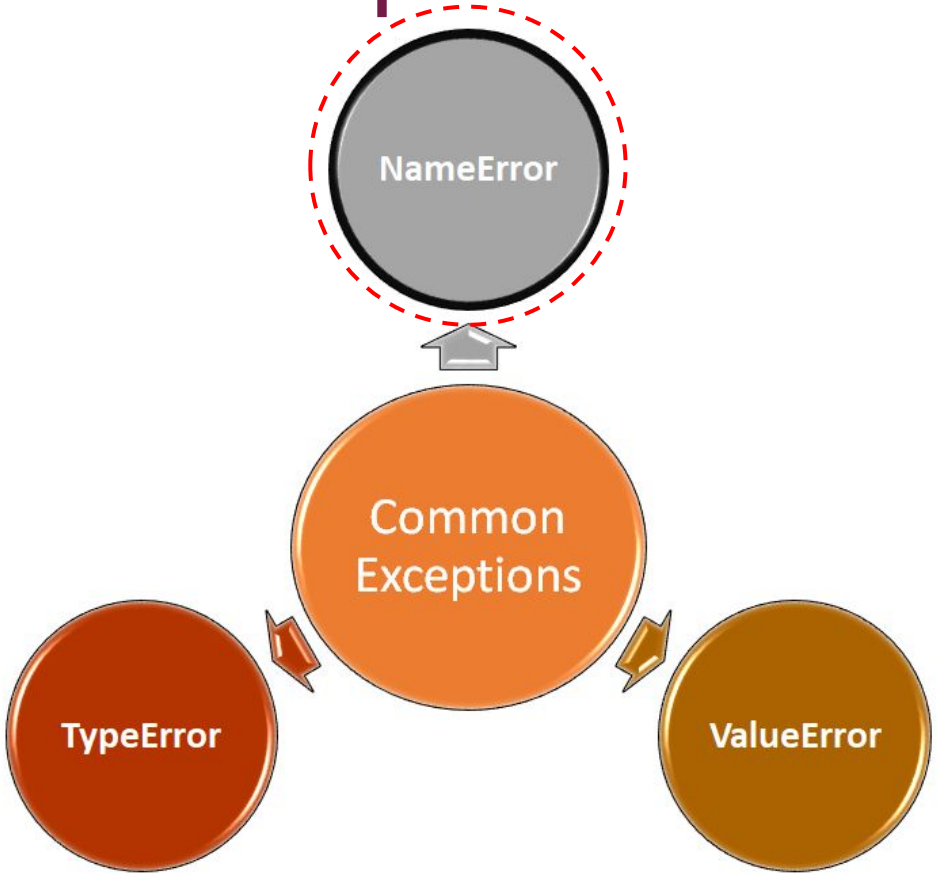
```
1  Traceback (most recent call last):
2    File "code.py", line 1, in <module>
3      print(int('ten'))
4  ValueError: invalid literal for int() with base 10: 'ten'
5
```

ValueError



Common Exceptions (review)

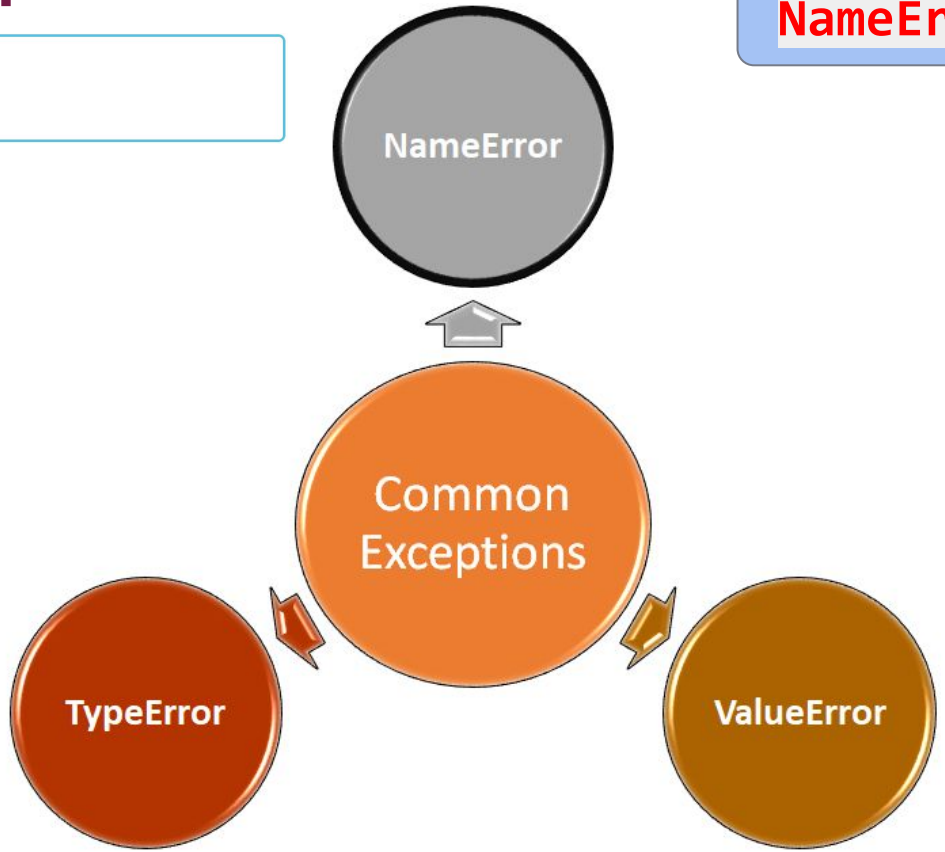
```
NameError
```



Common Exceptions (review)

```
1  print(variable)
2  variable = "Don't ever give up!"
```

NameError

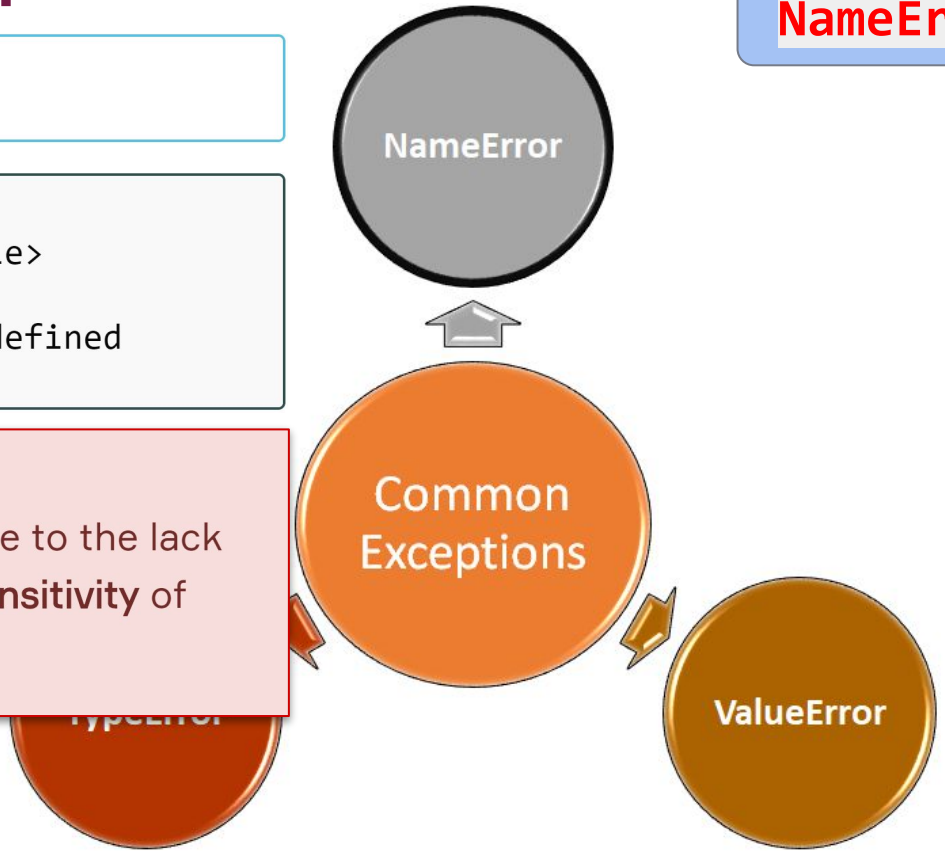


Common Exceptions (review)

NameError

```
1  print(variable)
2  variable = "Don't ever give up!"
```

```
1  Traceback (most recent call last):
2    File "code.py", line 1, in <module>
3      print(variable)
4  NameError: name 'variable' is not defined
5
```

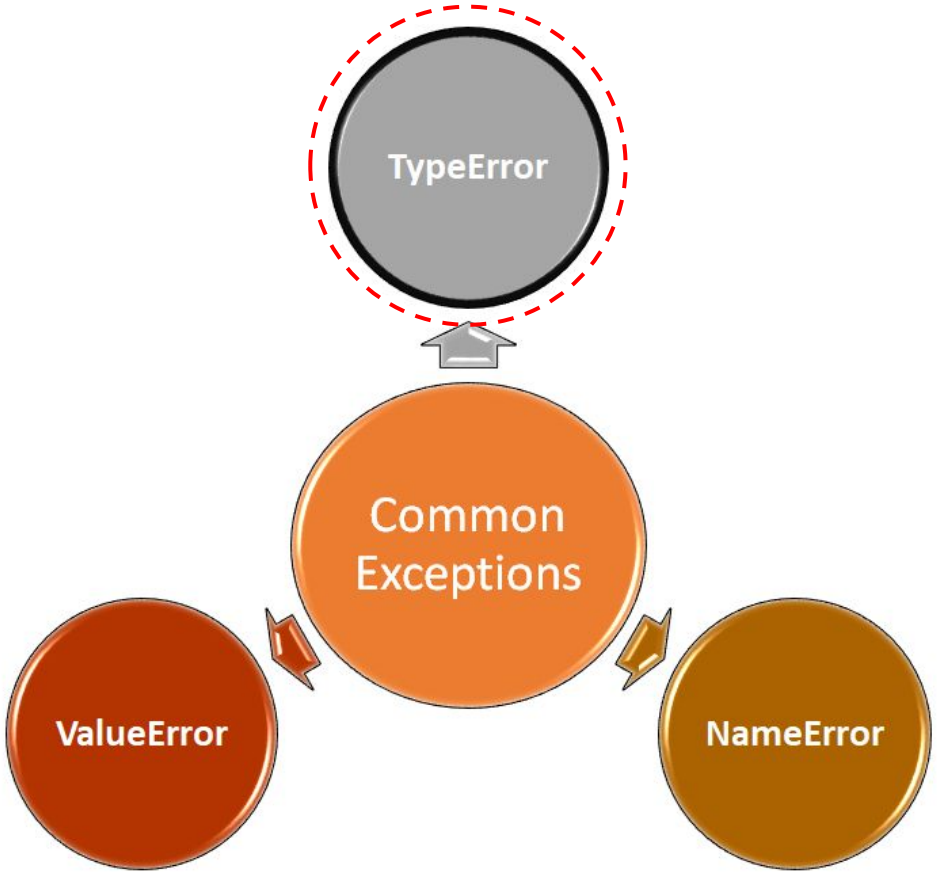


⚠ **Attention :**

- Note that the NameError often raises due to the lack of attention to these two things: **case-sensitivity** of Python and **pre-defines** of the variables.

Common Exceptions (review)

```
TypeError
```

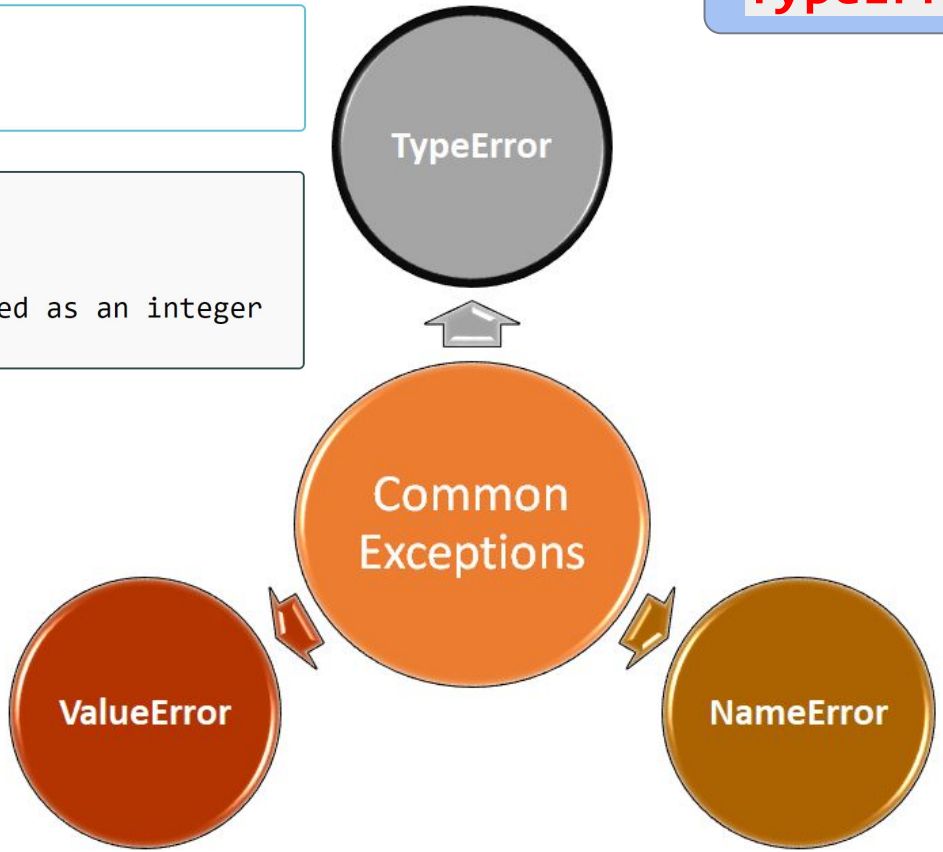


Common Exceptions (review)

```
1  for i in range('x'):
2      print(i)
3
```

```
1  Traceback (most recent call last):
2    File "code.py", line 1, in <module>
3      for i in range('x'):
4  TypeError: 'str' object cannot be interpreted as an integer
5
```

TypeError



Common Exceptions (review)

Tips:

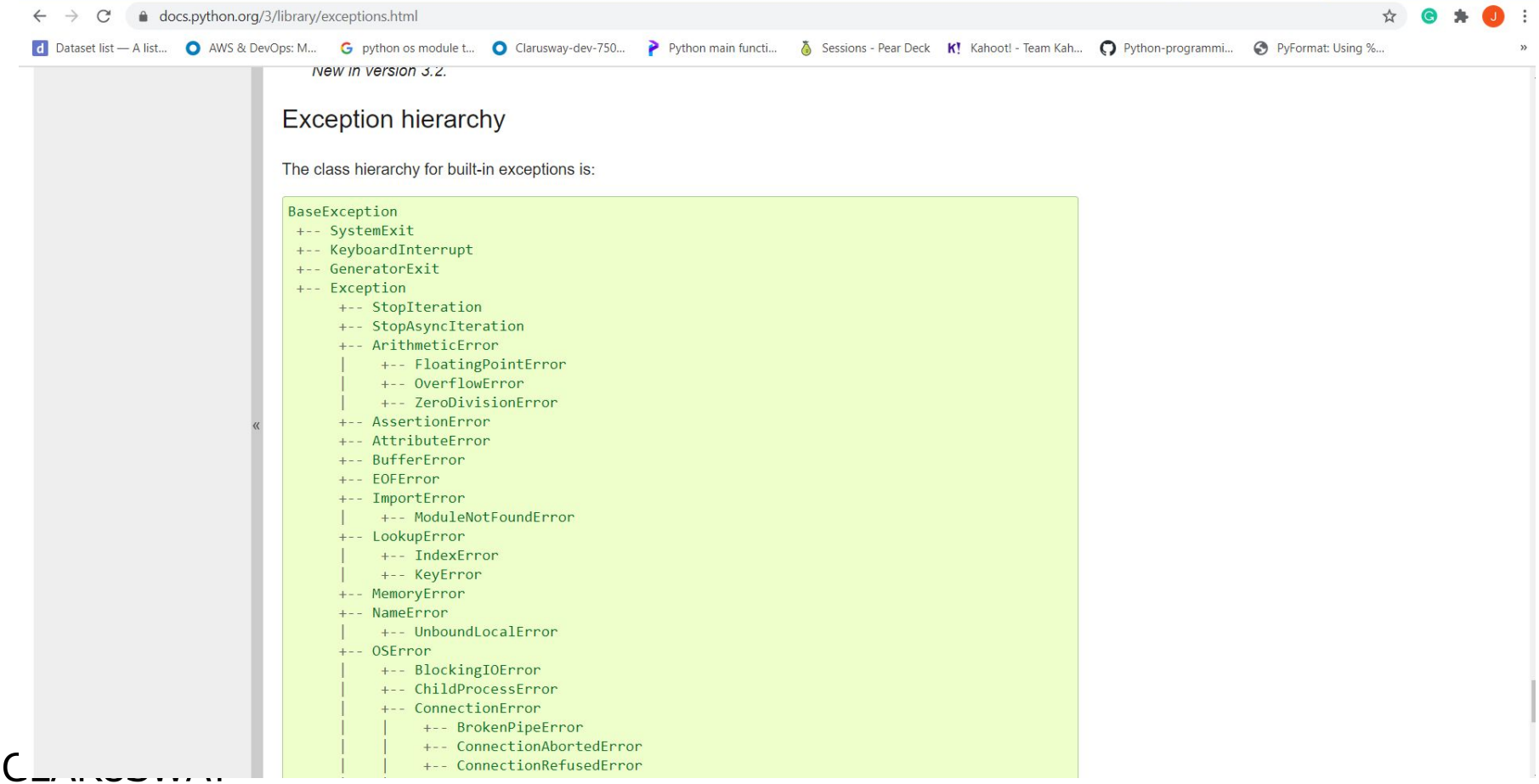
- The most useful way you can do about all the errors you can't deal with yourself is to search for the error message on the internet search engine.
- You can make sure that the errors that you will encounter and their solutions have been experienced by someone previously.

Exception Handling

Table of Contents

1 Introduction

Introduction (review)



Introduction (review)

- Example :

```
1  while True:
2      no_one = int(input("The first number please : "))
3      no_two = int(input("The second number please : "))
4      division = no_one / no_two
5      print("The result of the division is : ", division)
6      break
```

Introduction (review)

- Let's see what happens if we input **4** and **0**.

```
1  while True:
2      no_one = int(input("The first number please : "))
3      no_two = int(input("The second number please : "))
4      division = no_one / no_two
5      print("The result of the division is : ", division)
6      break
```

```
1  First number please : 4
2  Second number please : 0
3  ---------------------------------------------------------------------------
4  Traceback (most recent call last)
5  <ipython-input-5-537f57020b40> in <module>
6        2     no_one = int(input("First number please : "))
7        3     no_two = int(input("Second number please : "))
8  ----> 4     division = no_one / no_two
9        5     print("The result of the divison is : ", division)
10
11 ZeroDivisionError: division by zero
```

Introduction (review)

- The basic structure of `try-except` **statement** looks like :

```
try:
    code block to be normally executed
except:
    code block to be exceptionally executed
```

Introduction (review)

- Let's fix this exception problem using `try-except` **statement** as follows and pass the numbers **5** and **0** :

```
1  while True:
2      no_one = int(input("The first number please : "))
3      no_two = int(input("The second number please : "))
4      try:
5          division = no_one / no_two
6          print("The result of the division is : ", division)
7          break
8      except:
9          print("Something went wrong...Try again.")
10
```

```
1  The first number please : 5
2  The second number please : 0
3  Something went wrong...Try again.
```

executes **except** block

Introduction (review)

- This time let's specify the exception type which is `ZeroDivisionError` :

```
1  while True:
2      no_one = int(input("The first number please : "))
3      no_two = int(input("The second number please : "))
4      try:
5          division = no_one / no_two
6          print("The result of the division is : ", division)
7          break
8      except ZeroDivisionError:
9          print("You can't divide by zero! Try again.")
10
```

We specify the **exception error**

Tips:

- Of course, when do this, we anticipate which exception error we will encounter.

Introduction (review)

- Let's pass the same numbers 5 and 0.:

```
1  while True:
2      no_one = int(input("The first number please : "))
3      no_two = int(input("The second number please : "))
4      try:
5          division = no_one / no_two
6          print("The result of the division is : ", division)
7          break
8      except ZeroDivisionError:
9          print("You can't divide by zero! Try again.")
10
```

```
1  The first number please : 5
2  The second number please : 0
3  You can't divide by zero! Try again.
4
```

Exception Handling

- **Task :**
  - Try to open a non-existing file named **`my_file.txt`** to read using **`try - except`** block,
  - Give the following message in case of exception raises:
    - `"There is not such a file or the path is incorrect."`

Exception Handling

- The **`try - except`** code block can be as follows **:**

```
1  try:
2      file = open("my_file.txt", "r")
3      print(file.read())
4  except:
5      print("There is not such a file or the path is incorrect.")
6
```

Output

```
There is not such a file or the path is incorrect.
```

Exception Handling

- **Task :**
  - Try to open a non-existing file named **`my_file.txt`** to read using **`try - except`** block,
  - Specify the **exception type** for except block,
  - Give the following message in case of exception raises:
    - `"There is not such a file or the path is incorrect."`

Exception Handling

- The **try - except** code block can be as follows:

```
1 try:
2     file = open("my_file.txt", "r")
3     print(file.read())
4 except FileNotFoundError:
5     print("There is not such a file or the path is incorrect.")
6
```

We specify the **exception error**

Output

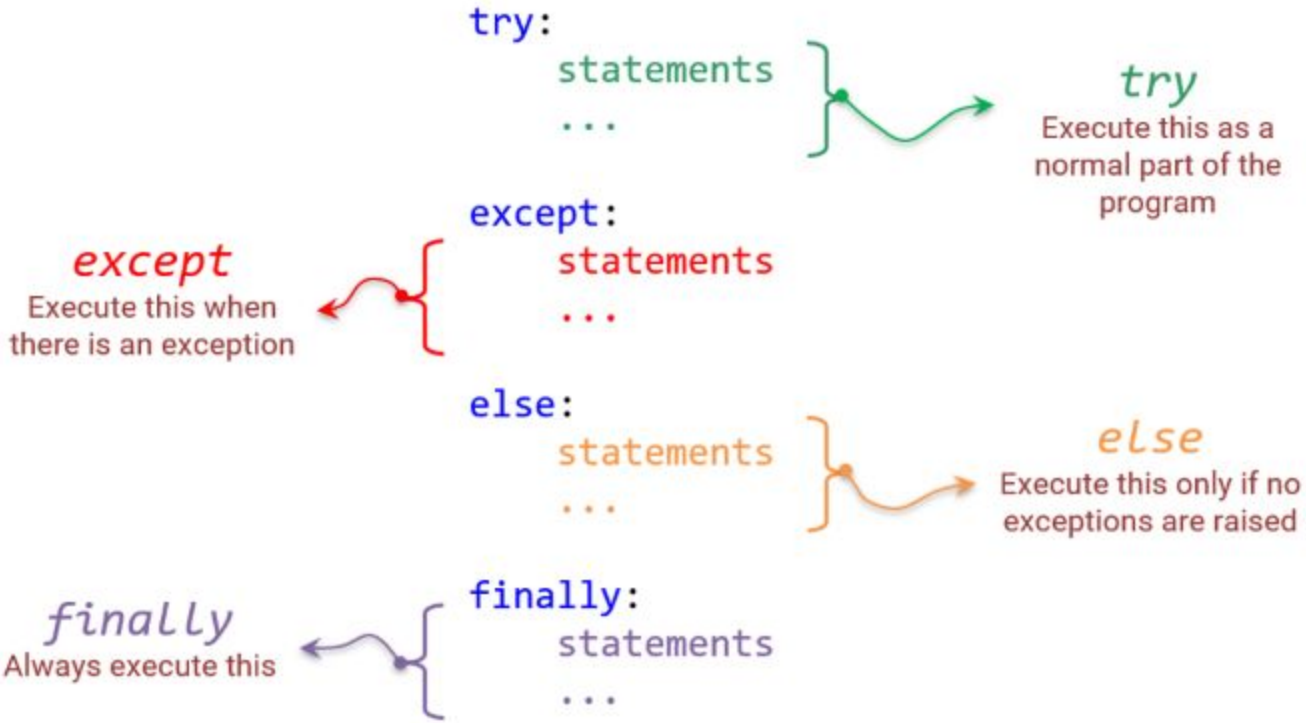
```
There is not such a file or the path is incorrect.
```

2

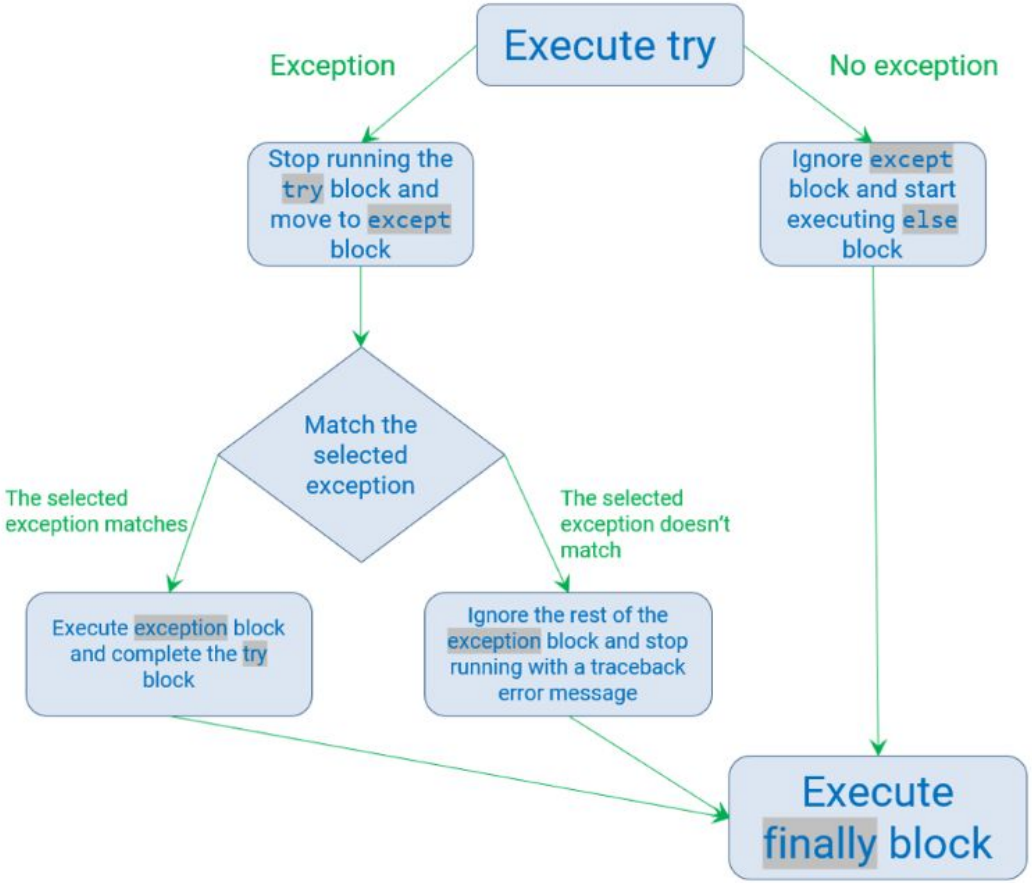
Full 'Exception Handling Block'

Full 'Exception Handling Block' (review)

- The full version of `try-except` **statement** is as follows :



Full 'Exception Handling Block' (review)



3

Implementation of the Full 'Exception Handling Block'

Implementation of the Full 'Exception Handling Block' (review)

```
while True:
    no_one = int(input("The first number please : "))
    no_two = int(input("The second number please : "))
    try:
        division = no_one / no_two  # normal part of the program
    except ZeroDivisionError:
        print("You can't divide by zero! Try again.")  # executes when
            division by zero
    else:
        print("The result of the division is : ", division)  # executes if
            there is no exception
    finally:
        print("Thanks for using our mini divison calculator! Come again!")
        break  # exits the while loop

```

```
The first number please : 7
The second number please : 2
The result of the division is :  3.5
Thanks for using our mini divison calculator! Come again!
```

Implementation of the Full 'Exception Handling Block' (review)

```
1  while True:
2      no_one = int(input("The first number please : "))
3      no_two = int(input("The second number please : "))
4      try:
5          division = no_one / no_two  # normal part of the program
6      except ZeroDivisionError:
7          print("You can't divide by zero! Try again.")  # executes when
               division by zero
8      else:
9          print("The result of the division is : ", division)  # executes if
               there is no exception
10     finally:
11         print("Thanks for using our mini divison calculator! Come again!")
12         break  # exits the while loop
13
```

input ⇒ 7 and 0

Implementation of the Full 'Exception Handling Block' (review)

```
while True:
    no_one = int(input("The first number please : "))
    no_two = int(input("The second number please : "))
    try:
        division = no_one / no_two  # normal part of the program
    except ZeroDivisionError:
        print("You can't divide by zero! Try again.")  # executes when
            division by zero
    else:
        print("The result of the division is : ", division)  # executes if
            there is no exception
    finally:
        print("Thanks for using our mini divison calculator! Come again!")
        break  # exits the while loop

```

```
The first number please : 7
The second number please : 0
You can't divide by zero! Try again.
Thanks for using our mini divison calculator! Come again!

```

Implementation of the Full 'Exception Handling Block' (review)

- This time, let's enter **`str`** type as a second number. The selected exception (**`ZeroDivisionError`**) will not match, so **`traceback error`** will be raised.

```
input ⇒ 7 and "k"
```

Implementation of the Full 'Exception Handling Block' (review)

- Let's see the result :

```
The first number please : 7
The second number please : k
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-2-5fd297d1c219> in <module>
      1 while True:
      2     no_one = int(input("The first number please : "))
----> 3     no_two = int(input("The second number please : "))
      4     try:
      5         division = no_one / no_two  # normal part of the program

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

```

Implementation of the Full 'Exception Handling Block' (review)

```
1  while True:
2      no_one = int(input("The first number please : "))
3      no_two = int(input("The second number please : "))
4      try:
5          division = no_one / no_two
6          print("The result of the division is : ", division)
7          break
8      except Exception as e:
9          print("Something went wrong...Try again.")
10         print("Probably it is because of '{}' error".format(e))
11         break
12
```

```
1  The first number please : 4
2  The second number please : 0
3  Something went wrong...Try again.
4  Probably it is because of 'division by zero' error.
5
```

Implementation of the Full 'Exception Handling Block' (review)

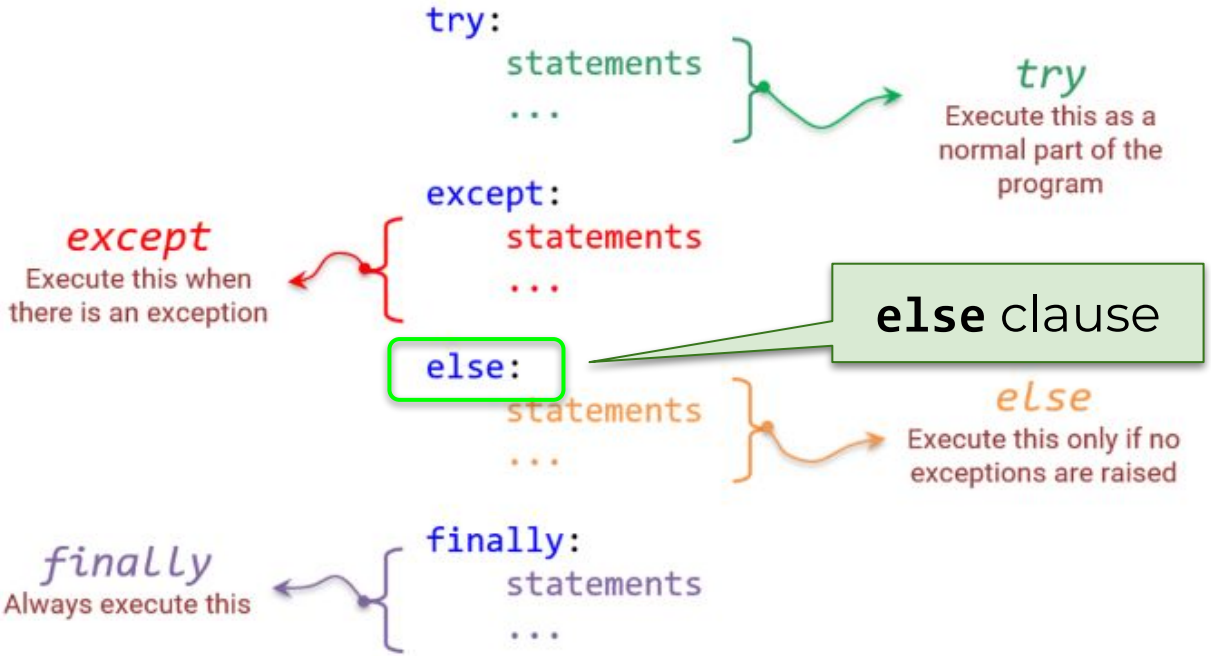
Exception as

```
1  try :
2      a = 10
3      b = 2
4      print("The result of division is :", c)
5  except Exception as e:
6      print("The error message is : ", e)
7
```

```
1  The error message is :  name 'c' is not defined
2
```

Implementation of the Full 'Exception Handling Block' (review)

`else`



Implementation of the Full 'Exception Handling Block' (review)

- Let's see the result :

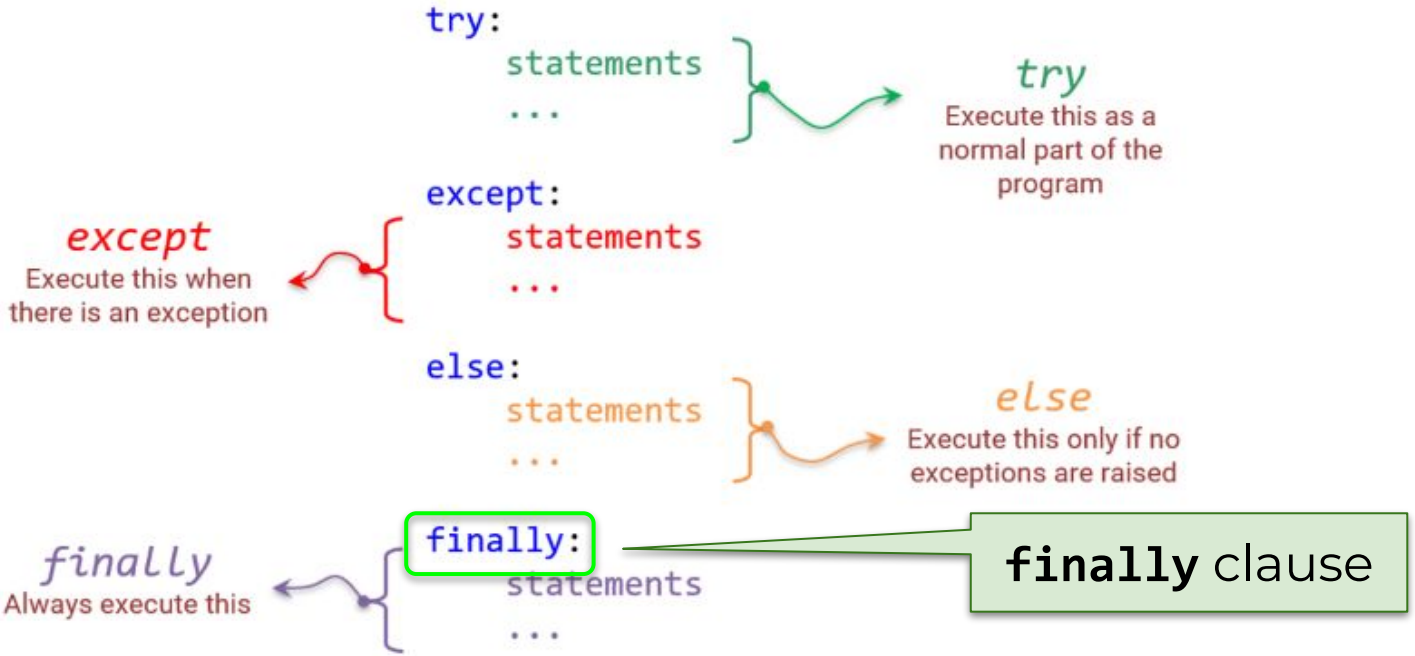
```
1  try:
2      x = 4 / 1
3  except:
4      print('Something went wrong')
5  else:
6      print('Nothing went wrong')
7
```

```
1  Nothing went wrong
2
```

else clause executes, because there is no exception

Implementation of the Full 'Exception Handling Block' (review)

finally



Implementation of the Full 'Exception Handling Block' (review)

- Let's see the result :

```
1  try:
2      x = 3 / 0
3  except:
4      print('Something went wrong')
5  finally:
6      print('Always execute this')
7
```

Although there is an `ZeroDivisionError` exception, `finally` block executes

```
1  Something went wrong
2  Always execute this
3
```

comes from **except**

comes from **finally**

4 Several Handling Scenarios

Several Handling Scenarios (review)

- In the example below you can define as many except blocks as you want, to catch and handle specific `exceptions` :

```
1  try:
2      x = 2/0
3  except ZeroDivisionError:
4      print('Attempt to divide by zero')
5  except:
6      print('Something else went wrong')
7
```

What is the output? Try to figure out in your mind...

Several Handling Scenarios (review)

- Here is the output :

```
1  try:
2      x = 2/0
3  except ZeroDivisionError:
4      print('Attempt to divide by zero')
5  except:
6      print('Something else went wrong')
7
```

first **exception** possibility

rest of all **exception** possibilities

```
1  Attempt to divide by zero
2
```

Several Handling Scenarios (review)

- Since all the built-in exceptions are formed a hierarchical structure, you can use the following exception syntaxes.
- **scenario-1:** you can identify no specific exception :

```
....
except:
    print("Unknown exception error occured. Tyr again please.")
```

- **scenario-2:** different exception blocks one after another :

```
....
except ZeroDivisionError:
   print("You can't divide by zero!!")
except ValueError:
   print("You can only enter numbers consisting of digits, not text!!")

```

Several Handling Scenarios (review)

- **scenario-3:** an except syntax also may contain multiple exceptions separated by commas and enclosed by parenthesis.

```
....
except (ValueError, TypeError):
    print("You can only enter numbers consisting of digits, not text!!")

```

- **scenario-4:** because of the hierarchical structure, one `exception` can actually catch **various exceptions**.

```
....
except ArithmeticError:
    print("I will also catch OverflowError, FloatingPointError and
        ZeroDivisionError")

```

Exception Handling

- **Task :**
  - Using `while` loop and `try ... except` block,
  - To select the favorite fruit from the given `fruits list`, write a program that requests an **index number** from the user and prints the selected fruit on the screen (**eg:** `"My favorite fruit is mango"`).
  - Considering the possible exceptions caused by the user's entry, display warning messages, and continue to ask for an **index number** from the user **until valid** input is taken.

```
fruits = ["banana", "mango", "pear", "apple", "kiwi", "grape"]
```

Exception Handling

- The code snippet can be as follows :

```
1  fruits = ["banana", "mango", "pear", "apple", "kiwi", "grape"]
2
3  while True:
4      try:
5          index = int(input("Pick an index number to choose your favorite fruit"))
6          print("My favorite fruit is", fruits[index])
7          break
8
9      except IndexError:
10         print("There is no such an index. Try again!")
11
12     except ValueError:
13         print("You should enter integer. Try again!")
14
```

Exception Handling

- **Task :**
  - Using `while` loop and **full** `try ... except` block,
  - At this time, in addition to the previous program,
  - Give the user **three chances** to enter an input, and whenever the user raises an exception, **decrease** the counter **by 1** displaying that `"<warning message>, you have n right left. Try again!"`
  - If no exception raise, display a message : `"Congrats! You've entered a valid input."`
  - In all circumstances, display a message : `"Our fruits are always fresh!"`

example

input ⇒ 1, output ⇒

```
My favorite fruit is mango
Congrats! You've entered a valid input.
Our fruits are always fresh!
```

Exception Handling

- The code snippet can be as follows :

```
fruits = ["banana", "mango", "pear", "apple", "kiwi", "grape"]
count = 3
while count>0:
    try:
        index = int(input("Pick an index number to choose your favorite fruit"))
        print("My favorite fruit is", fruits[index])
    except IndexError:
        count -= 1
        print(f"There is no such an index. You have {count} right left. Try again!")
    except ValueError:
        count -= 1
        print(f"You should enter integer. You have {count} right left. Try again!")
    else:
        print("Congrats! You've entered a valid input.")
        break
    finally:
        print("Our fruits are always fresh!")

```