



# Exception Handling





# Table of Contents

- ▶ Introduction
- ▶ Full 'Exception Handling Block'
- ▶ Implementation of the Full 'Exception Handling Block'
- ▶ Several Handling Scenarios



1

# Introduction

# Did you find this pre-class interesting and challenging?



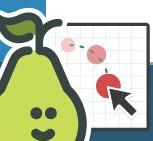
Too hard



Just right



Too easy



Students, drag the icon!



Pear Deck Interactive Slide  
Do not remove this bar

# Introduction (review)



docs.python.org/3/library/exceptions.html



Dataset list — A list...

AWS & DevOps: M...

python os module t...

Clarusway-dev-750...

Python main functi...

Sessions - Pear Deck

Kahoot! - Team Kah...

Python-programmi...

PyFormat: Using %...

»

NEW IN VERSION 3.2.

## Exception hierarchy

The class hierarchy for built-in exceptions is:

```
BaseException
+-- SystemExit
+-- KeyboardInterrupt
+-- GeneratorExit
+-- Exception
    +-- StopIteration
    +-- StopAsyncIteration
    +-- ArithmeticError
        |    +-- FloatingPointError
        |    +-- OverflowError
        |    +-- ZeroDivisionError
    +-- AssertionError
    +-- AttributeError
    +-- BufferError
    +-- EOFError
    +-- ImportError
        |    +-- ModuleNotFoundError
    +-- LookupError
        |    +-- IndexError
        |    +-- KeyError
    +-- MemoryError
    +-- NameError
        |    +-- UnboundLocalError
    +-- OSError
        |    +-- BlockingIOError
        |    +-- ChildProcessError
        |    +-- ConnectionError
            |        +-- BrokenPipeError
            |        +-- ConnectionAbortedError
            |        +-- ConnectionRefusedError
```

# 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.

```
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 The first number please : 6  
2 The second number please : 3  
3 The result of the division is : 2.0
```



# Introduction (review)

- ▶ Let's see what happens.

```
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 The first number please : 6  
2 The second number please : 3  
3 The result of the division is : 2.0
```

```
1 The first number please : 7  
2 The second number please : 2  
3 The result of the division is : 3.5
```

# 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
```

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



USWY<sup>®</sup>  
Students, write your response!

REINVENT YOURSELF

Pear Deck Interactive Slide  
Do not remove this bar

# 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
```

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



USMAY®  
REINVENT YOURSELF



# 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
```

executes  
**except** block

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



# 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.
```

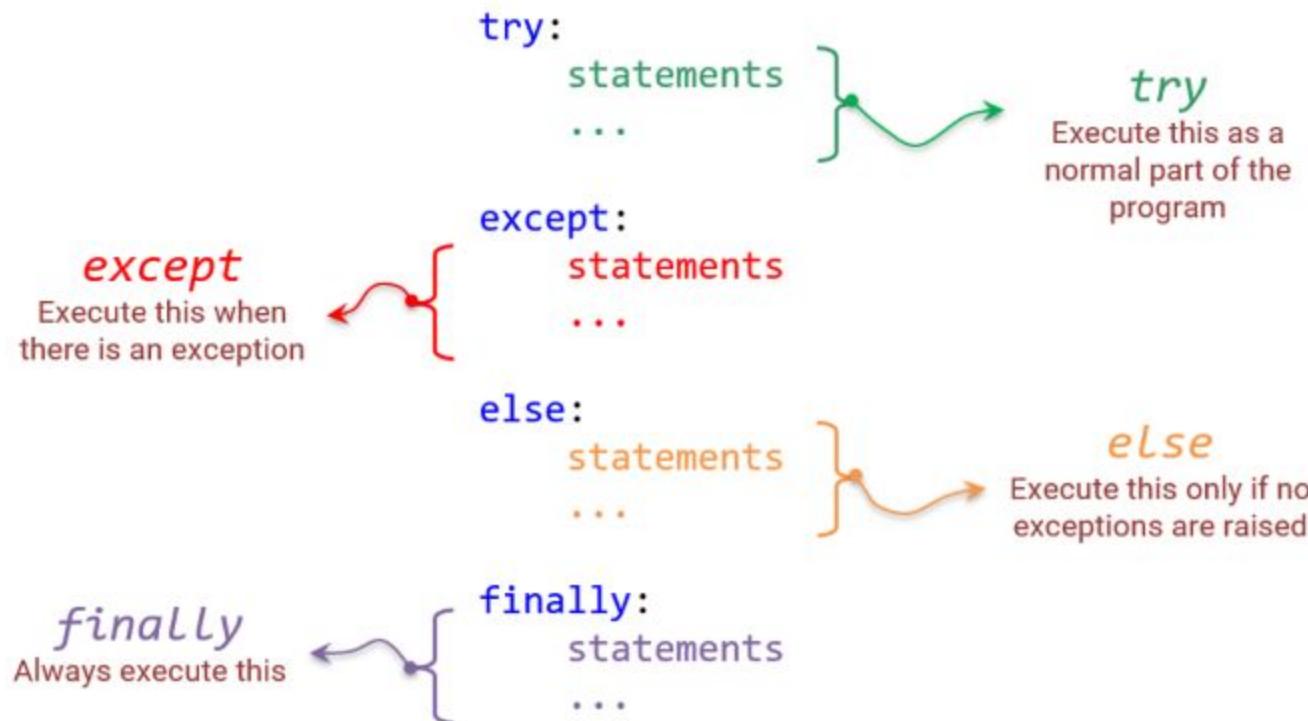


2

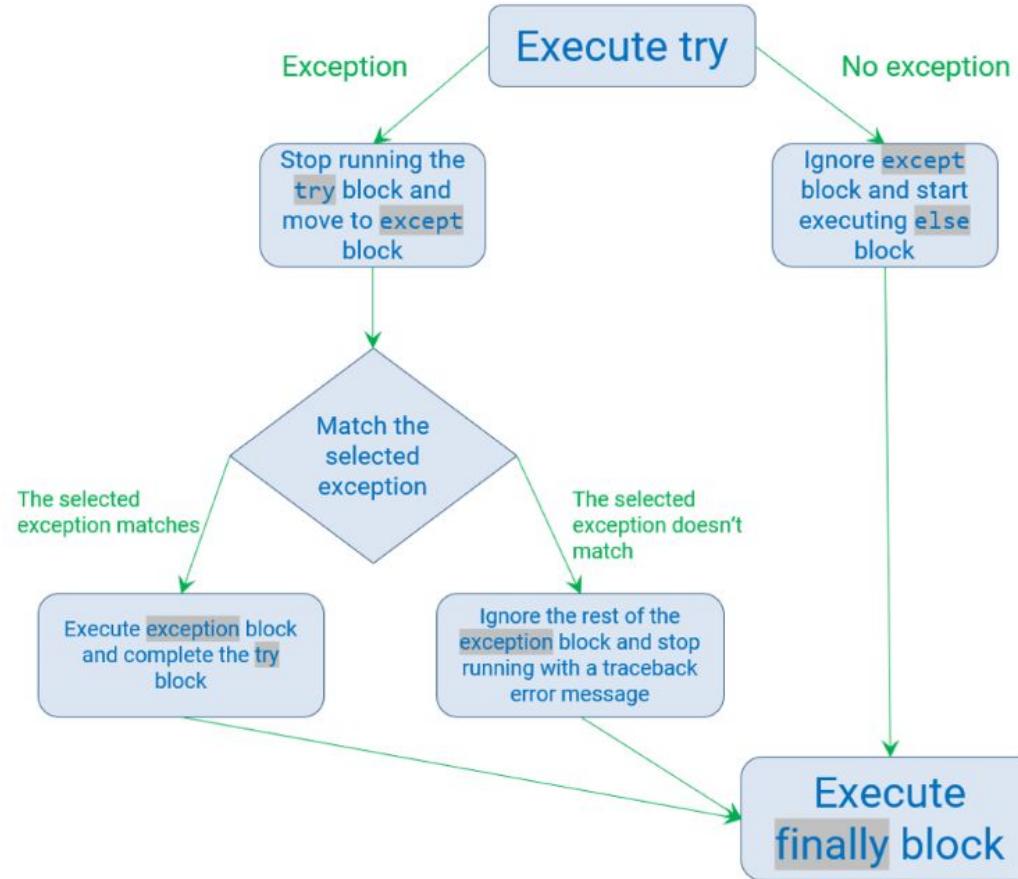
## Full 'Exception Handling Block'

# ► 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)

```
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  
8             division by zero  
9     else:  
10        print("The result of the division is : ", division) # executes if  
11            there is no exception  
12    finally:  
13        print("Thanks for using our mini divisor calculator! Come again!")  
14        break # exits the while loop
```

input  $\Rightarrow$  7 and 2

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



Students, write your response!

REINVENT YOURSELF



# 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  
8             division by zero  
9     else:  
10        print("The result of the division is : ", division) # executes if  
11            there is no exception  
12    finally:  
13        print("Thanks for using our mini divisor calculator! Come again!")  
14        break # exits the while loop
```

```
1 The first number please : 7  
2 The second number please : 2  
3 The result of the division is : 3.5  
4 Thanks for using our mini divisor 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  
8             division by zero  
9     else:  
10        print("The result of the division is : ", division) # executes if  
11            there is no exception  
12    finally:  
13        print("Thanks for using our mini divisor calculator! Come again!")  
14        break # exits the while loop
```

input ⇒ 7 and 0



# 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  
8             division by zero  
9     else:  
10        print("The result of the division is : ", division) # executes if  
11            there is no exception  
12    finally:  
13        print("Thanks for using our mini divisor calculator! Come again!")  
14        break # exits the while loop
```

```
1 The first number please : 7  
2 The second number please : 0  
3 You can't divide by zero! Try again.  
4 Thanks for using our mini divisor calculator! Come again!  
5
```



# 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 :

```
1 The first number please : 7
2 The second number please : k
3 -----
4 ValueError                                     Traceback (most recent call last)
5 <ipython-input-2-5fd297d1c219> in <module>
6     1 while True:
7         2     no_one = int(input("The first number please : "))
8     ----> 3     no_two = int(input("The second number please : "))
9         4     try:
10             5         division = no_one / no_two # normal part of the program
11
12 ValueError: invalid literal for int() with base 10: 'k'
13
```

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

- By using `Exception` keyword, we can display the name of the exception type.

```
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
```

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



input  $\Rightarrow$  4 and 0

Students, write your response!

# 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)

- Yes, as you can see the output, by using the keyword **Exception**, we can display the name of the exception type. Consider the other example :

```
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
```

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



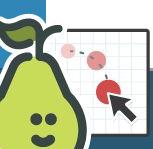
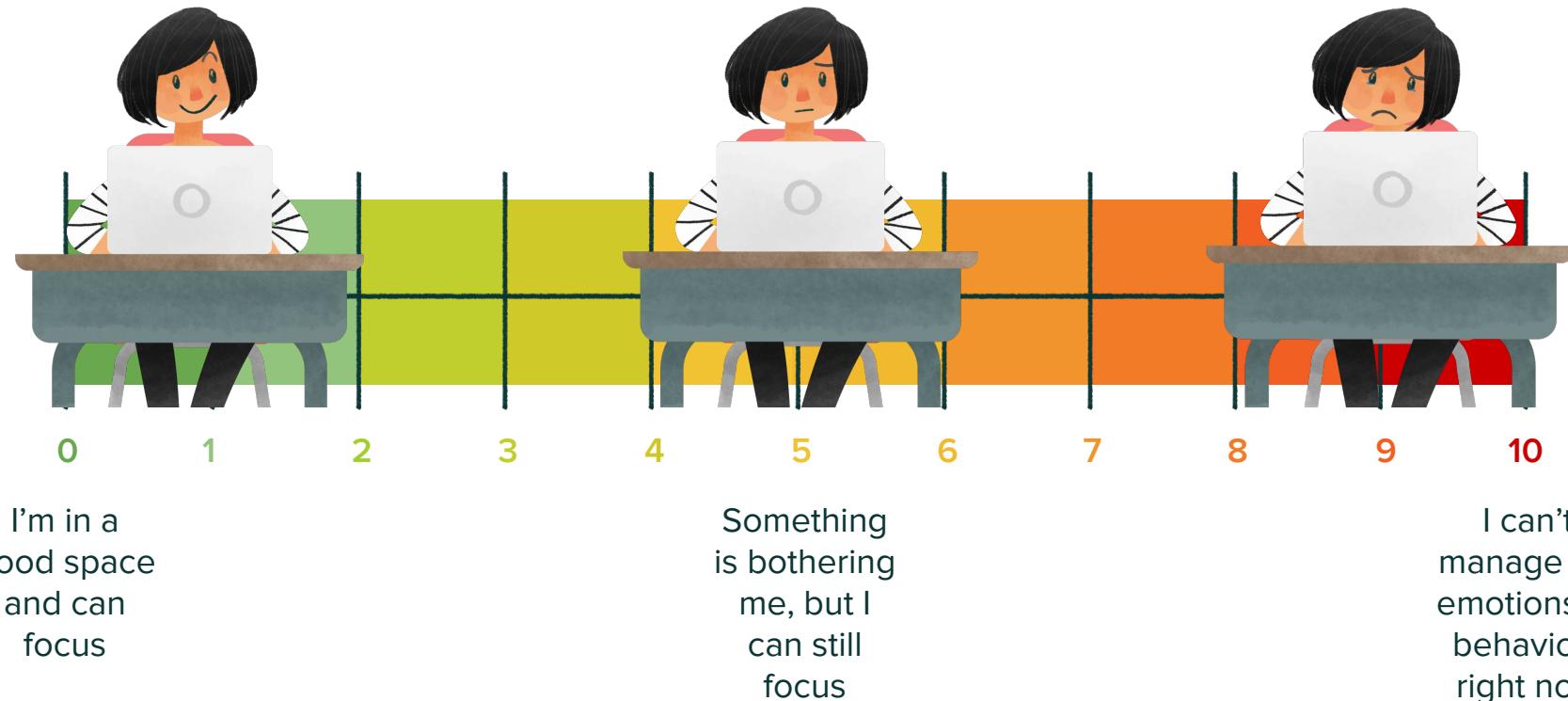
# Implementation of the Full 'Exception Handling Block' (review)

- Yes, as you can see the output, by using the keyword **Exception**, we can display the name of the exception type. Consider the other example :

```
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
```

# Stress Check

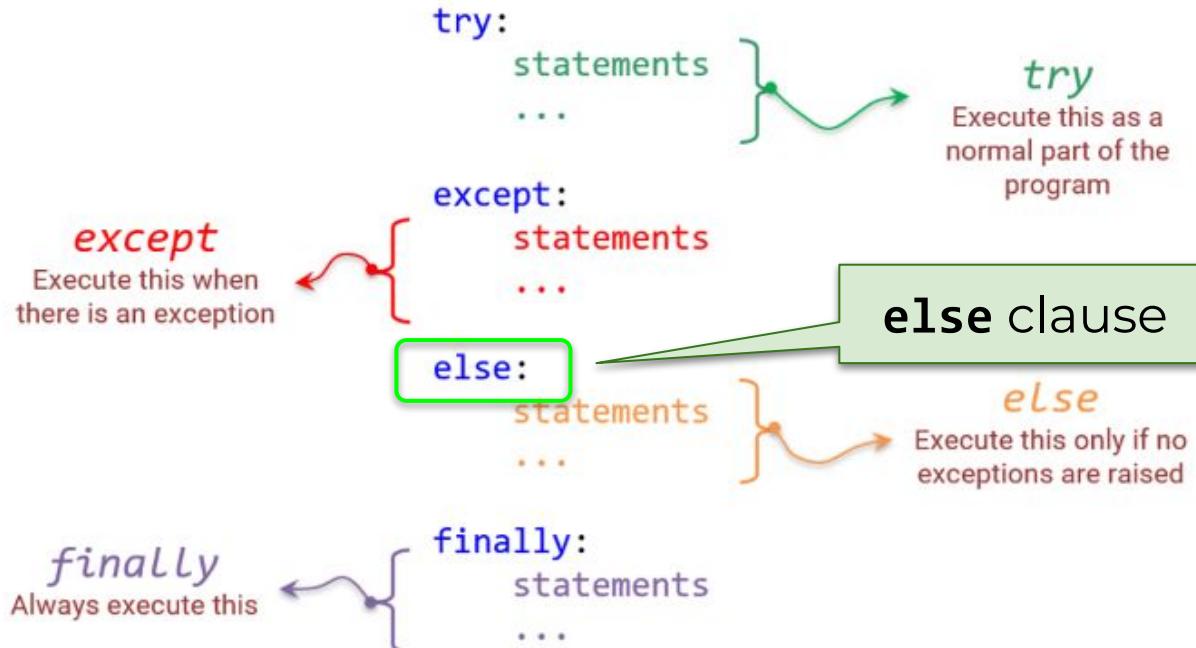


Students, drag the icon!



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

- The **try...except** block has an optional **else** clause. The **else** clause is executed only if no exceptions are raised.



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

- ▶ Consider the following example of the `else` clause :

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



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

- Let's see the result :

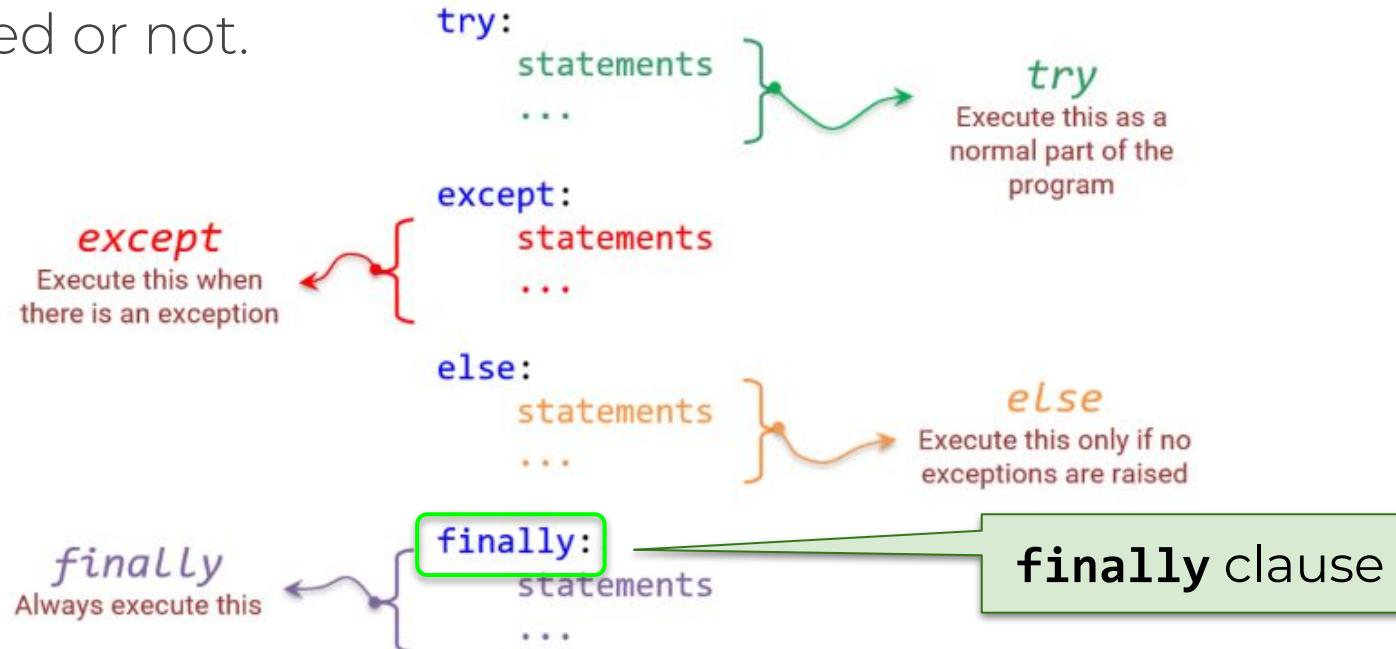
```
1 try:  
2     x = 4 / 1  
3 except:  
4     print('Something went wrong')  
5 else:  
6     print('Nothing went wrong')  
7
```

```
1 Nothing went wrong  
2
```

**else** clause executes, because there is no exception

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

- The **try...except** block has another optional **finally** clause. The finally clause is always executed, whether an exception has occurred or not.





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

- ▶ Consider the following example of the `finally` clause :

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

# 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**

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

- ▶ Use `finally` clause to define **clean-up actions** that must be executed under all circumstances e.g. closing a file. :

```
1 try:  
2     f = open('myfile.txt')  
3     print(f.read())  
4 except:  
5     print("Something went wrong")  
6 finally:  
7     f.close()  
8
```



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
```

```
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.
- ▶ **input-1:** you can identify no specific exception :

```
1 |....  
2 ▾ except:  
3     print("Unknown exception error occurred. Try again please.")
```

- ▶ **input-2:** different exception blocks one after another :

```
1 |....  
2 ▾ except ZeroDivisionError:  
3     print("You can't divide by zero!!")  
4 ▾ except ValueError:  
5     print("You can only enter numbers consisting of digits, not text!!")  
6
```

# ► Several Handling Scenarios (review)

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

```
1 |....  
2 - except (ValueError, TypeError):  
3     print("You can only enter numbers consisting of digits, not text!!")  
4
```

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

```
1 |....  
2 - except ArithmeticError:  
3     print("I will also catch OverflowError, FloatingPointError and  
         ZeroDivisionError")  
4
```

# 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  $\Rightarrow$  1, output  
 $\Rightarrow$

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 :

```
1 fruits = ["banana", "mango", "pear", "apple", "kiwi", "grape"]
2 count = 3
3 while count>0:
4     try:
5         index = int(input("Pick an index number to choose your favorite fruit"))
6         print("My favorite fruit is", fruits[index])
7     except IndexError:
8         count -= 1
9         print(f"There is no such an index. You have {count} right left. Try again!")
10    except ValueError:
11        count -= 1
12        print(f"You should enter integer. You have {count} right left. Try again!")
13    else:
14        print("Congrats! You've entered a valid input.")
15        break
16    finally:
17        print("Our fruits are always fresh!")
18
```



# Well Done!

You can describe yourself  
as “has Intermediate  
Python Skill”

